UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Máster Universitario en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones

Extensión Semántica de OML (Semantic-OML)

Mario Poyato Pino

septiembre de 2014

Extensión Semántica de OML (Semantic-OML)

AUTOR: Mario Poyato Pino

TUTOR: Jorge E. López de Vergara

Grupo: High Performance Computing & Networking (HPCN)

Dpto. Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

septiembre de 2014

Resumen

OpenLab es un proyecto europeo dentro del séptimo programa marco (FP7) que pretende proporcionar un conjunto bancos de pruebas para realizar experimentos en redes a gran escala. Los bancos de pruebas incluyen equipamiento de red y herramientas para obtener medidas de retardos y pérdidas de paquetes, uso de ancho de banda de enlace, seguimiento de paquetes, etc.

Sin embargo, cada herramienta de monitorización obtiene, clasifica y organiza las medidas de la red de forma diferente. En la mayoría de los casos estas medidas proporcionan información similar pero estructurada de forma distinta, lo que hace necesario definir y utilizar la ontología MOI definida por la ETSI de forma que se homogenice el esquema conceptual.

Por ello, el objeto de este Trabajo Fin de Master es, partiendo de la ontología MOI, plantear la extensión y mejora de la biblioteca OML, utilizada en los bancos de prueba para recopilar las medidas de red. En OML se definen los denominados puntos de medida, que con la extensión desarrollada almacenarán las medidas en bases de datos semánticas mejorando de esta forma el rendimiento y estandarización de la ontología.

Abstract

OpenLab is a European project of the Seventh Framework Program (FP7) that aims to provide a set of testbeds to perform experiments in large-scale networks. The testbeds include network equipment and tools to measure delays and packet losses, link usage bandwidth, package tracking, etc.

However, each monitoring tool obtains, classifies and organizes the network measurements differently. In most cases these measurements provide similar information but structured in other way, making it necessary to define and use the MOI ontology defined by ETSI so the conceptual schema is homogenized.

Therefore, the aim of this Master Thesis is, based on the MOI ontology, to provide the extension and improvement of the OML library, used in the testbeds to collect the network measurements. In OML, the so-called measuring points are defined, which will store the measurements in semantic databases with the developed extension, thus improving performance and standardization of the MOI ontology.

Agradecimientos

Quisiera agradecer en primera instancia a mis padres y a mi familia que sin su apoyo durante la realización de este máster y la carrera me han permitido concentrarme durante todos estos años en la realización de mis tareas en la universidad.

También a Sandra por su apoyo en todo momento y hacerme en el último fragmento de mi estancia en la universidad el mejor de todos a pesar de la carga de este trabajo ha supuesto en mi vida permitiéndome conciliarlo con mi vida laboral.

Posteriormente mencionar a mis compañeros de la carrera y del máster con quienes he compartido una fase importante en mi vida con quienes he materializado todas las tareas y necesidades propuestas desde mi ingreso en la universidad.

Sin olvidar a mis compañeros del laboratorio, a Rafael, a Diego, a Pedro, a Víctor… de quienes he aprendido muchas cosas y me han ayudado en momentos que he necesitado dándome no solo apoyo técnico sino una gran amistad.

También me siento agradecido con miembros del laboratorio como Sergio, Iván, Gustavo o Javier Aracil que aunque no he tratado demasiado con ellos en mis tareas todos han sido profesores y grandes mentores a los que me siento orgulloso de haber tenido.

Y por último y no por ello menos importante, a Jorge, que ha tutelado toda mi estancia en el laboratorio desde mis prácticas hasta en este trabajo final de máster. Que me ha ayudado en todos los problemas que han surgido de forma instantánea y siempre ha estado proporcionándome todo lo que necesitaba aunque yo no supiera de su necesidad.

A todos, gracias.

Índice de Contenidos

Índice de Figuras

1 Introducción

1.1 Motivación

En la actualidad debido a la proliferación de las redes en nuestro entorno se hace necesario caracterizarlas mediante el uso de herramientas e instrumentos de medición. Como consecuencia, surgió el proyecto OpenLab [1], que provee a la comunidad científica de un conjunto de bancos de datos de medidas de red que permiten obtener propiedades como rendimiento, retardos…

Sin embargo, cada herramienta de monitorización obtiene, clasifica y organiza las medidas de la red de forma diferente. En la mayoría de los casos estas medidas proporcionan información similar pero estructurada de forma distinta, lo que hace necesario definir y utilizar una ontología que homogenice todos los conceptos y relaciones entre estos [2] [3].

En este caso se ha utilizado la ontología *Measurement Ontology for IP traffic* (MOI) definida por la *European Telecommunications Standards Institute* (ETSI) [4][5][6] para la medición sobre redes IP. Partiendo de aquí se diseñó una solución directa que era desplegar diversos servidores D2R que traduzcan los datos de las bases de datos relacionales al campo semántico. El principal problema de esta solución es que consume una gran cantidad de recursos y memoria porque la realización de la correspondencia entre tablas y conceptos de la ontología no es directa y requiere en algunos casos realizar productos de tablas o funciones almacenadas SQL para obtener el valor asociado a un concepto de la ontología.Por todo ello, el objeto de este proyecto es extender OMF Measurement Library (OML), el principal entorno de recolección de medidas de red de OpenLab para que directamente inserte las medidas obtenidas en bases de datos semánticas según la ontología de ETSI MOI en lugar de bases de datos relacionales que luego precisan de servidores D2R.

1.2 Objetivos

El objetivo principal, como se ha indicado más atrás, es añadir una capa semántica al framework de recolección de medidas OML que permita trabajar directamente con bases de datos semánticas, eliminando el coste computacional de mantener un servicio que interpreta por cada consulta una correspondencia concepto ↔ campo de base datos para poder generar la correspondiente consulta SQL.

En concreto, este objetivo se ha dividido en los siguientes subobjetivos:

- Definición y validación de un esquema flexible que cumpla la estructura de conceptos definidos en la ontología MOI definida por la ETSI [4][5][6].
- Soporte de los conceptos y estructura semántica definida por el programador tanto para procesamiento de los puntos de medida (MPs) en el cliente y en el servidor OML a la vez que la transmisión entre estos.
- Generación de las sentencias de inserción en el servidor de OML que soporte el estándar SPARQL/Update [9] y transmisión de estas para su inserción en el endpoint.
- Validación de la estructura semántica definida para los MPs con la ontología.

1.3 Metodología y plan de trabajo

Para alcanzar los objetivos propuestos, se ha establecido la siguiente metodología de trabajo:

1. Estudio del Arte
   1.1. Bases de datos semánticas.
      1.1.1. Investigación y comprensión de la estructura de la ontología MOI definida por la ETSI [4][5][6].
      1.1.2. Investigación de implementaciones de BBDD semánticas.
         1.1.2.1. Utilización de SPQR y D2R [12].
         1.1.2.2. Análisis de Fuseki [10], Open Virtuoso [11], Parliament [13].
   1.2. Estructura actual de actuación para bases de datos reportadas por el framework OML.
      1.2.1. Estudio del funcionamiento del código fuente de OML.
      1.2.2. Estudio de la articulación interna de los mappings necesarios para los servidores D2R.
      1.2.3. Montaje y ejecución de los servidores D2R.
2. Desarrollo de la capa semántica sobre la biblioteca de recolección de medidas OML.
   2.1. Generar el código necesario para definir la estructura semántica.
   2.2. Soportar el reporte de la estructura conceptual cliente/servidor OML.
   2.3. Realización de inserciones unitarias y respetando el estándar SPARQL/Update 1.1 [9].
   2.4. Añadir las validaciones necesarias para que el programador sepa si la estructura semántica definida es correcta.
   2.5. Mostrar sugerencias de corrección a nivel de código y ejecución del programa OML/cliente en caso de que la estructura propuesta sea incorrecta.
3. Pruebas funcionales.
   3.1. Validación de los resultados y comprobación de que se obtienen los mismos resultados a nivel funcional que la solución D2R+SQL que directamente una base de datos SPARQL.
4. Pruebas de rendimiento.
   4.1. Observar y medir la perdida de rendimiento en la inserción de la capa semántica respecto a la relacional.
   4.2. Observar y medir la ganancia de rendimiento en las consultas de un endpoint semántico al de una base de datos relacional con la unión de un servidor D2R.
5. Evaluación de la satisfacción del trabajo realizado.
   5.1. Medir hasta qué punto se han cumplido las expectativas del proyecto.
6. Realización de la documentación y la memoria del proyecto.

1.4 Organización del documento

La memoria se estructura como sigue, a continuación en el capítulo 2 se indica el contexto actual de este proyecto, se introduce el concepto de ontología y la estandarización de la ETSI con su ontología para medidas de red. Más adelante, dentro de este mismo capítulo se presenta el framework OML y la solución actual a la conversión de las base de datos relacionales producidas por este framework en base de datos semánticas. Posteriormente, se finaliza el capítulo realizando un análisis de los endpoints semánticos existentes en el mercado.

Tras estudiar el estado del arte, se indica cómo se ha realizado la ampliación semántica de la biblioteca OML en el capítulo 3, con la siguiente validación de los resultados y análisis del rendimiento en el capítulo 4. Finalmente, se concluye el documento y se muestra el posible trabajo futuro el capítulo 5.

2 Estado del arte

2.1 Introducción

Tal y como se indicó en la motivación, en la mayoría de los casos las herramientas de obtención de medidas de red proporcionan información similar pero estructurada de distinta manera, lo que hace necesario definir y utilizar un mecanismo que homogenice todos los conceptos y relaciones entre estos [2] [3]. El mecanismo propuesto en este trabajo es una ontología de conceptos de medidas de red la cual se definirá y se comentarán algunas peculiaridades de esta solución más adelante en la sección 2.2.

La ontología que se utiliza para homogenizar los datos es la ontología MOI definida por la ETSI [4][5][6] para la medición sobre redes IP. Esta ontología contiene la mayoría de los conceptos necesarios para tratar medidas de redes IP y las relaciones entre estos. La cual esta descrita en mayor profundidad en el apartado 2.3.

Más adelante, se introduce la primera solución a la homogenización del conjunto de los bancos de datos. Dónde en primera instancia se optó por la solución más directa que utiliza la ontología descrita en la sección 2.4.

Pero debido a los requerimientos computacionales y rendimiento en las consultas, esta solución no es óptima por lo que se propone convertir el esquema de las bases de datos relacionales a un esquema semántico y para ello se utiliza el framework OML [7], un framework que reporta los resultados de los experimentos en bases de datos relacionales generadas al vuelo. OML se detallará más en la sección 2.5.

Este Trabajo Fin de Máster se realiza en el contexto del proyecto OpenLab [1], el cual proporciona un conjunto de bancos de prueba de algunas de las principales redes internaciones. Dichos bancos de prueba serán introducidos con más detalle en la sección 2.6. El alto uso de la librería OML en diversos testbeds de OpenLab hace necesario estudiar los beneficios de generar bases de datos semánticas en lugar de relacionales debido a la naturaleza de la estandarización. Lo que propicia realizar un estudio de las diversas soluciones de base de datos semánticas, el cual será planteado en el apartado 2.7.

2.2 Ontologías

Las ontologías informáticas son, en esencia, clasificaciones conceptuales utilizadas como medio para categorizar y agrupar información en conceptos y reglas de correspondencia que junto a la reutilización de definiciones de conceptos generales como son W3CTime, FOAF, Units… las hacen idóneas para estandarizaciones y homogenización de datos llevándolos al plano semántico.

Al principio esta estructura se utilizaba en inteligencia artificial y sistemas expertos, posteriormente en agentes inteligentes y web semántica y actualmente en cualquier aplicación que requiera establecer un modelo de información, que es nuestro caso.

A efectos prácticos estas estructuras semánticas tienen las ventajas de trabajar con los conceptos y sus relaciones con otros conceptos lo que da lugar a una mejor organización de la información y por tanto una optimización de su definición.

Además en la actualidad se puede aprovechar desarrollos realizados en el ámbito de la web semántica como la edición de ontologías donde utilizamos Protegé [17] o bibliotecas de manipulación y servidores de consultas como puede ser el D2R Server [12].

En conclusión, se trata de una *especificación explícita y formal de una conceptualización compartida* [43] que cumple con las siguientes características:

- *Explícita*: compuesta de conceptos, propiedades, relaciones, funciones, axiomas y restricciones
- *Formal*: puede ser interpretada por máquinas
- *Conceptualización*: modelo abstracto del dominio a representar
- *Compartida*: acordada por grupos de expertos

2.2.1 Definición de una ontología: RDF, RDFS y OWL

RDF [44] es un lenguaje definido por el W3C que permite realizar un modelado conceptual parecido a los diagramas de clases pero con la idea principal de definir todas las relaciones conceptuales a modo de triplas sujeto, objeto y predicado. Una extensión de este lenguaje es RDFS [45] que añade sobre el anterior algunas relaciones básicas que dan soporte a la definición de una ontología, como el tipo de un concepto o el tipo y relaciones de una clase. Sobre la definición de RDFS el W3C ha definido OWL, el cual añade conceptos y relaciones nuevas como el concepto de ancestro o similitud.

Debido a su naturaleza ambos lenguajes pueden ser especificados con un fichero con formato XML, turtle [16]…

En el caso de la definición de la ontología de ETSI MOI se ha utilizado el programa Protegé [17] con una interfaz gráfica que permite definir todos los conceptos como clases y sus relaciones como conexiones entre clases permitiendo almacenar toda la definición en formato OWL.

2.2.2 Lenguaje de consultas: SPARQL 1.1

SPARQL 1.1 [30] [31] es un estándar de facto que define la sintaxis y la semántica del lenguaje de consultas para endpoints semánticos. Este modelo se trata de un lenguaje con bastante similitud a SQL, la diferencia principal es que SQL supone que los datos están implementados en tablas y SPARQL/Update supone que los datos están implementados en grafos. Además define además de diversos términos como IRI [32], o identificador de una instancia RDF, una sintaxis con el fin de poder realizar consultas para la recuperación de datos de diversas fuentes siempre que los datos se almacenan de forma nativa semántica o son definidos mediante vistas RDF a través de algún sistema middleware como puede ser el servidor D2R del que se hablará más adelante.

SPARQL permite la consulta de los patrones obligatorios y opcionales de un grafo de conceptos, junto con sus conjunciones y disyunciones además de soportar la ampliación o restricción del ámbito de las consultas indicando los grafos sobre los que se opera. Los resultados de las consultas SPARQL pueden ser tanto datos o estructura de grafos semánticos.

Un ejemplo de una consulta SPARQL se puede encontrar en la figura de más adelante, que obtiene los RTT de cada medida de traceroute con un TTL inicial menor que una constante, por ejemplo 20.

```
PREFIX MD: <http://www.etsi.org/isg/moi/data.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?traceroute ?rtt {
    ?traceroute a MD:Traceroute ;
        MD:hasMeasurementData ?a .
    ?a a MD:RoundTripDelayMeasurement ;
        MD:RoundTripDelayMeasurementValue ?rtt .
    FILTER (?rtt < "20"^^xsd:long )
}
LIMIT 100
```

Figura 2-1: Consulta SPARQL de ejemplo

Como se puede ver en la propia consulta SPARQL tiene una sintaxis parecida a SQL donde en el SELECT realiza una proyección sobre los datos obtenidos y el contenido de dentro de las llaves impone las restricciones que se deben cumplir los datos obtenidos, aunque este lenguaje soporta realizar ordenación, agregación, asignaciones…

Algunos de los elementos básicos del lenguaje son:

- PREFIX: declaración del espacio conceptual.
- SELECT: realiza una proyección de los resultados
- DESCRIBE: devuelve un grafo que describe los recursos encontrados.
- ASK: indica si la combinación sujeto-verbo-predicado existe en la ontología RDF.
- FROM: indica los elementos sobre los que se ejecutará la consulta, en caso de no definirse tomará todos los grafos definidos en el endpoint.
- ORDER BY: es opcional y permite ordenar los resultados de la consulta.
- LIMIT Y OFFSET: permite seleccionar un subconjunto de los resultados.
- GROUP BY: realiza agregación en los resultados.

2.2.3 Lenguaje de inserciones: SPARQL/Update 1.1 sobre HTTP.

SPARQL/Update es una extensión de SPARQL que permite además de consultar modificar ontologías, permitiendo la creación, modificación y borrado de triplas para un grafo determinado. Un ejemplo de una consulta SPARQL/Update 1.1 es:

```
PREFIX MD: <http://www.etsi.org/isg/moi/data.owl#>

INSERT DATA {
       <http://example/iperf> a MD:Iperf ;
           MD:hasMettricAttributes <http://example/srcIP> .
       <http://example/srcIP> a MD:SourceIP ;
           MD:SourceIPValue "127.0.0.1"^^xsd:string .
}
```

Figura 2-2: Consulta SPARQL/Update de ejemplo

Cualquier servidor que implemente el protocolo SPARQL/Update 1.1. y recibe una petición acerca de un IRI que identifica el contenido del grafo RDF que debe ser absoluta de forma que identifique al grafo, aunque algunas implementaciones indiquen que si no se especifica un grafo se usa uno por omiso. La siguiente figura ilustra esto.

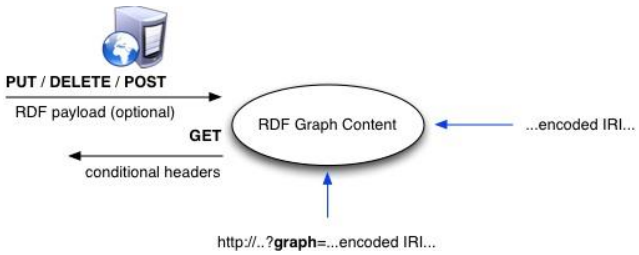


Figura 2-3: Diagrama que identifica la necesidad de especificar un punto único de inserción. [48]

Una extensión de esta especificación es SPARQL/Update 1.1 sobre HTTP actúa como una serialización de la comunicación en las operaciones SPARQL/Update 1.1 que utiliza el protocolo HTTP para realizar operaciones de actualización. La siguiente figura ilustra esta extensión.

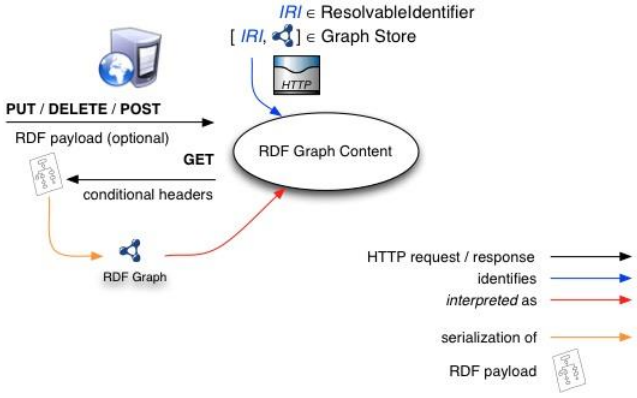


Figura 2-4: Un diagrama del modelo de protocolo para modificaciones directas del grafo semántico [48]

Aunque utiliza los métodos HTTP estándar, como GET, POST, PUT y DELETE, estos métodos no son suficientes por lo que es necesario añadir las clausulas INSERT, DELETE y MODIFY, LOAD, CLEAR, CREATE y DROP.

2.3 Ontología de ETSI MOI y estandarización

Como se ha indicado en el apartado anterior una ontología permite realizar una estandarización de los conceptos y reglas de asociación de esta. Algunas de las alternativas son SQL, dónde es necesario definir una vista común para las distintas bases de datos pero tiene el inconveniente de distribuir una consulta en los distintos modelos de SQL cómo son: SQLite, MySQL, PostgreSQL…, otra alternativa es utilizar XML aunque se queda únicamente en el aspecto sintáctico y XMLSchema permite restringir tipos de datos, pero aquí se trata más bien de especializar. Por lo que la mejor alternativa es utilizar una ontología que como se ha indicado en el apartado anterior se utilizan los conceptos y sus relaciones por lo que da una mayor formalización y conceptualización además de ser explicita y compartir gran parte de su conocimiento entre grupos de expertos como se indicó en la sección anterior.

Por consiguiente, se ha utilizado una ontología estandarizada por la ETSI, en concreto, el grupo de trabajo MOI. Esta ontología consta de diversas partes bien diferenciadas. Como son: primeramente, la parte de los conceptos generales en la que se encuentran los protocolos, localizaciones, marcas de tiempo…; posteriormente, unidades, que identifican la unidad de medida; los metadatos, que contienen la información acerca de qué, cuándo, dónde y cómo se han tomado las medidas; finalmente, contiene las medidas en sí mismas además de la estructura básica y más cercana a estas entre los diferentes conceptos. Se puede ver la relación entre las diferentes partes en la siguiente figura.

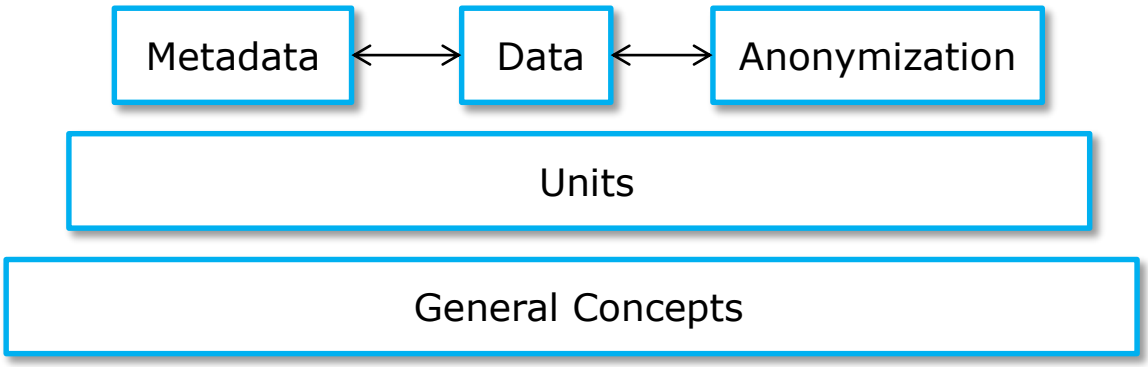


Figura 2-5: Estructura de la ontología

2.3.1 Conceptos generales

En detalle los conceptos generales son un conjunto básico de elementos que la ontología ETSI MOI debe contener desde el punto de vista de las mediciones de la red, como pueden llegar a ser la definición de circuito, paquete, red, conexión, dispositivo, y sus respectivas relaciones.

2.3.2 Unidades

Por debajo de los conceptos generales se sitúa la parte de unidades de Ontología ha sustituido todas las unidades del campo de la física generadas por la NASA [18] por las

unidades del conjunto de conocimiento de redes de computadores como bit, byte, bps, direcciones IP, etc. Teniendo en cuenta también el uso final de la ontología, que es la clasificación de las fuentes de datos heterogéneos, se ha diseñado teniendo en cuenta diferentes unidades para diferentes representaciones de direcciones de red. Las propiedades de transformación entre ellos no son sólo los factores numéricos como en las unidades del Sistema Internacional, también son expresiones regulares para igualar y llevar a cabo la transformación de la unidad, asegurando todas las conversiones entre unidades.

2.3.3 Metadatos

Por debajo de las unidades sección de los Metadatos de la Ontología describen la información sobre las medidas. Contiene información básica sobre la medición: descripción de los datos de medición, su tamaño, un hash de los contenidos (por razones de integridad de datos) y una descripción textual del proceso de creación, así como otra información importante de las fuentes de datos. El esquema básico de los metadatos se ha basado en DatCat (CAIDA) [19] en la que una clase a:DataMetadata relaciona los metadatos con las medidas como se muestra en la Figura 2-6.

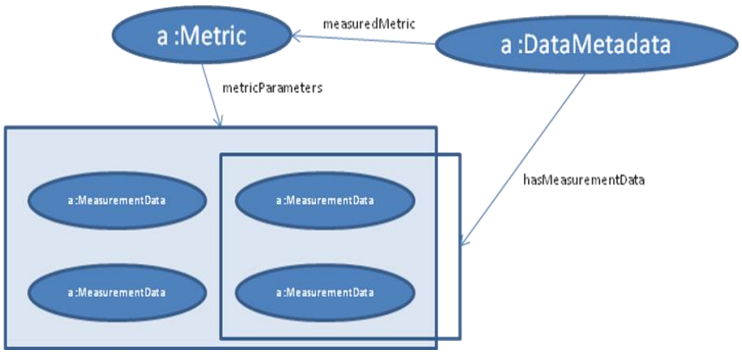


Figura 2-6: Estructura de la ontología [5]

2.3.4 Datos

Relacionada con los metadatos esta la ontología de datos comprende las clases y propiedades para poder representar a "cualquier" medida que se ha generado a partir de cualquier herramienta de medición, infraestructura o investigador. Por lo tanto, como se indica en [5], envuelve las mediciones que van desde simples archivos de texto a las medidas orientadas a objetos complejos. La estructura general de la ontología de datos es la que aparece reflejada en la siguiente figura.

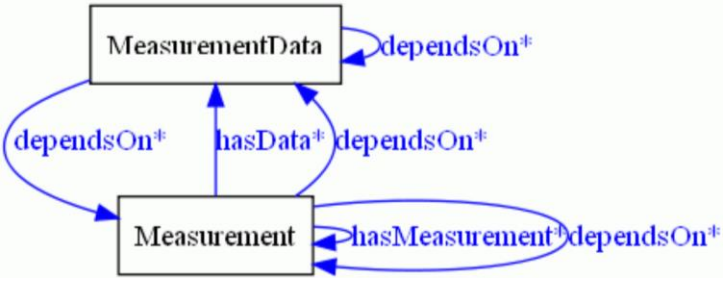


Figura 2-7: Estructura general de la ontología de datos. [5]

En concreto hay un conjunto de conceptos de tipo medida que pueden contener más medidas para finalmente tener medidas con datos que dependen de las anteriores. Con este esquema general se llega especializar al nivel mostrado en la Figura 2-8 más adelante. Donde una medida puede llegar a tener una métrica, la cual puede ser calculada de cierta forma.

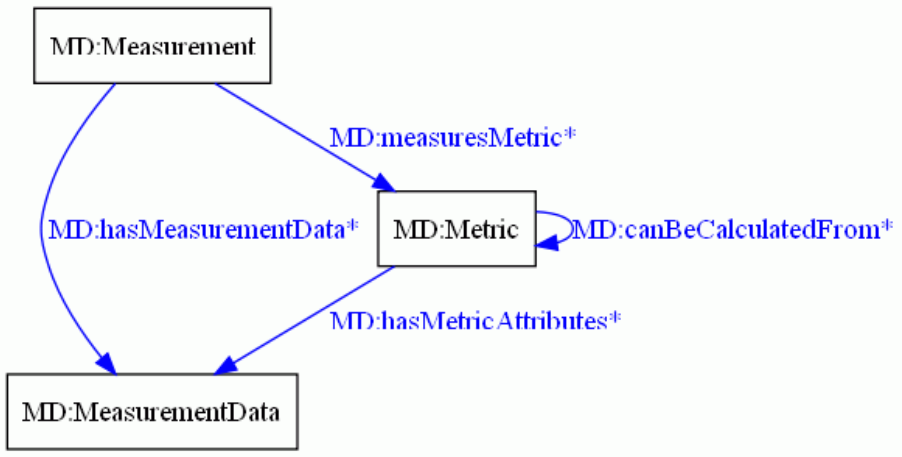


Figura 2-8: Estructura general de la ontología de datos [5]

Teniendo en cuenta todo lo anterior la estructura permite reglas de correspondencia bastante flexibles y a la que se suelen asociar en todos los mapping de bases de datos relacionales no semánticas como tabla con un concepto del tipo Measurement y columna con los datos de la propia medida MeasurementData.

2.3.5 Anonimización

Por último, el anonimato de la ontología representa el "vocabulario común" que deberá adaptarse a todos los posibles contextos de la anonimización, es decir, el conjunto mínimo de conceptos y / o funcionalidades que describe el dominio del "anonimato de las mediciones de red".

2.4 Conversión de bases de datos relacionales a semánticas

Como ya se vio en la sección 2.3 en este trabajo se ha utilizado la ontología ETSI MOI la cual provee de una base de conocimiento con proyecto de estándar y que permite una asignación de conceptos bastante generalista y flexible aproximación. Esta es asignar a la tabla el concepto de Measurement y a cada columna el concepto de MeasurementData y únicamente posteriormente establecer que métricas tiene cada columna.

Uno de los ejemplos proporcionados por OML es Iperf que es una aplicación que nos permite calcular el ancho de banda máximo disponible de una conexión, además de una tasa de errores por paquete… analizando el ejemplo de Iperf con la aproximación indicada al principio de esta sección se puede asignar los conceptos de SourceIP a la IP origen, DestinationIP a la IP destino y finalmente la capacidad disponible al ancho de banda disponible, pero además se puede añadir que tipo de dato es y en que unidad esta medido, se puede ver de forma esquemática en la siguiente figura.

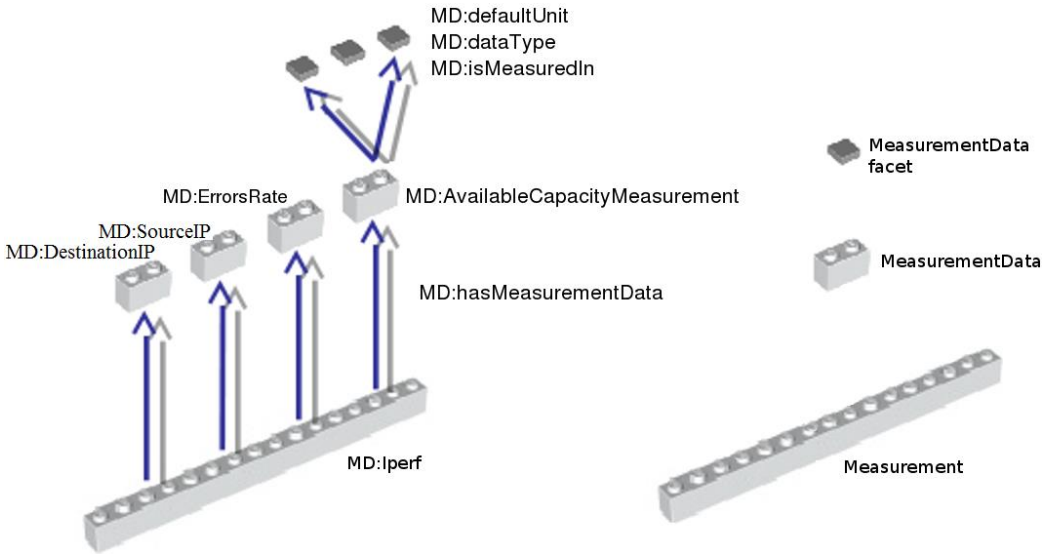


Figura 2-9: Mapping general para Iperf (OML).

Este mapping o asociación indicados se pueden materializar con la biblioteca D2R, cuyo funcionamiento aparece esquematizado en la siguiente figura donde a través de un fichero con las reglas de correspondencias entre la base de datos relacional (SQL) y los conceptos de la ontología se lanza un servicio que actúa como una interfaz semántica RDF sobre la que realizar consultas SPARQL. Estas reglas de correspondencia crean una asociación entre concepto y una columna, proyección u operación de una tabla.

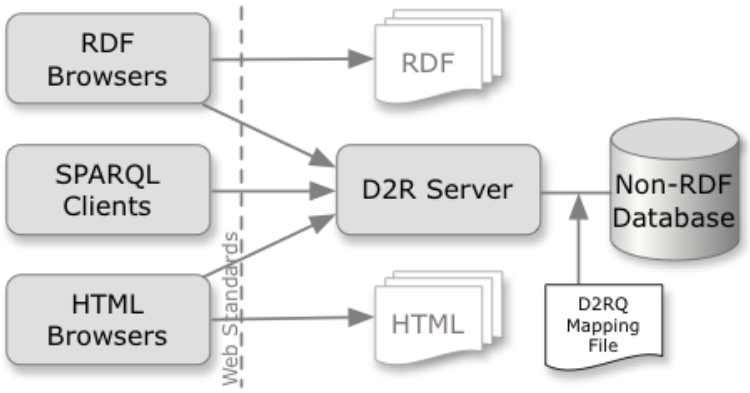


Figura 2-10: Esquema de funcionamiento de D2R. [12]

Por todo ello, el proceso de mapping de las bases de datos o conversión a base de datos semánticas de Openlab se realizaba de forma manual lo que obligaba a seguir una serie de pasos:

1. Comprender la estructura de la base de datos.

2. Utilizar la herramienta de generación automática de D2R [22].

3. Realizar la asociación de clases de la ontología con las tablas o expresiones SQL sencillas de tablas.

4. Levantar el servidor D2R con el fichero que contiene las asociaciones.

2.4.1 Mapping de ejemplo

De forma que si fijamos el subconjunto de la ontología de datos definido en la siguiente figura. Donde se tienen los conceptos Measurement y MeasurementData, siendo respectivamente el concepto de medida y medida de datos. Se podría, de esta forma, asociar conceptos como medida Ping, Iperf, Traceroute… al concepto Measurement (como medida) y los datos que miden estarían relacionados con MeasurementData (datos de medida).

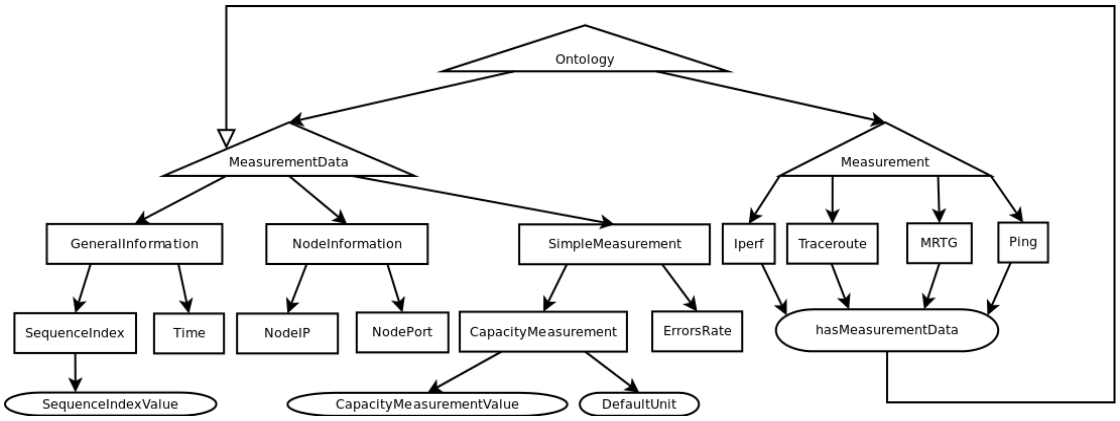


Figura 2-11: Estructura general de la ontología de datos.

Un ejemplo de mapping para la aplicación Iperf [23] que viene de ejemplo en el paquete de OML podría ser asignar a las columnas con las direcciones IP origen y IP destino los conceptos MD:NodeIP y a otra columna MD:CapacityMeasurement y todas ellas a su vez tener como dominio la clase MD:Iperf. Además de poder añadir la unidad en que esta medida la capacidad de la conexión y la unidad por defecto. De forma esquemática se puede visualizar en la siguiente figura.

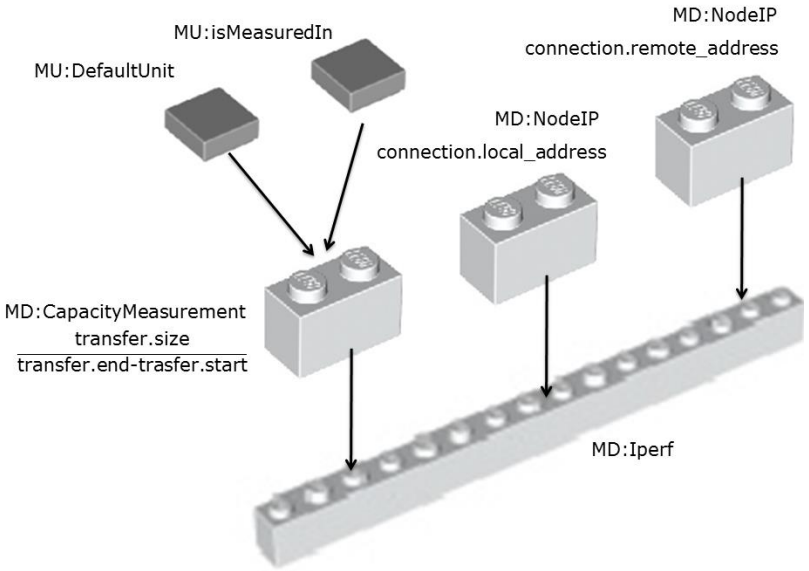


Figura 2-12: Estructura general de la ontología de datos.

Con lo que se ha cumplido con el objetivo de estandarizar la base de datos, pudiendo ahora consultar la capacidad máxima de una conexión dada o la conexión con más

capacidad sin preocuparse del esquema de la base de datos además de poder utilizar otros testbeds mapeados que tengan medidas IP pero que manejen los mismos conceptos.

Aunque se ha apreciado un gran detrimento en la velocidad de la consulta y un gran uso de la memoria debido a que el servidor D2R debe mantener las asociaciones. En algunos casos las consultas transformadas a SQL no son óptimas lo que hace pensar que pueda llegar a ser una solución bastante pesada. Se profundizará más en este debate en el apartado 4.4.

2.5 OML y su uso para recolección de medidas

2.5.1 Introducción

OML [7] es una herramienta de instrumentación que permite a los desarrolladores de aplicaciones definir puntos de medición configurables dentro de nuevas o pre-existentes aplicaciones y reportar los datos de las distintas mediciones de forma transparente a bases de datos SQL. Los experimentadores que ejecutan las aplicaciones pueden dirigir los flujos de mediciones desde estos puntos de medición a puntos de recolección remotos, para almacenar en bases de datos de medición.

OML consta de dos componentes bien diferenciadas:

- Biblioteca cliente/reportador OML: la biblioteca cliente OML proporciona una API C para aplicaciones que recogen medidas que producen u obtienen ya sea de la red o un dispositivo. La biblioteca incluye un mecanismo de filtrado configurable de la medición antes de reenviar al servidor OML. Al principio existía una implementación C aunque se ha extendido a Python (OML4Py) y Ruby (OML4R).

- Servidor/recolector OML: el componente de servidor OML se encarga de recoger y almacenar las mediciones dentro de una base de datos. Actualmente, SQLite3 y PostgreSQL se admiten como backends de bases de datos aunque en este trabajo se ha ampliado a un backend puramente semántico aunque eso se explicará más detalladamente más adelante.

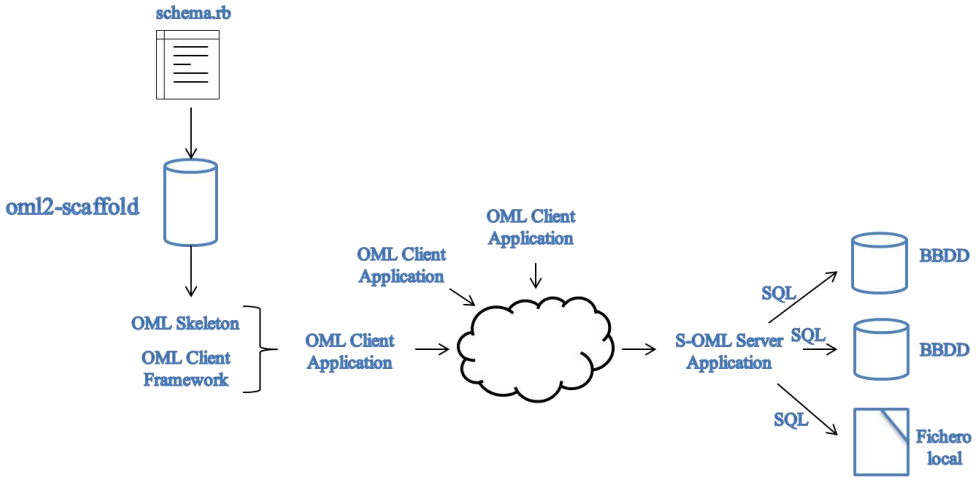


Figura 2-13: Diagrama de la arquitectura de OML.

2.5.2 OML Scaffold

Además de lo anterior la biblioteca contiene un asistente para la generación del cliente OML a partir de una definición conceptual en Ruby [20].

El uso de este asistente está recomendado para las últimas versiones, además de tal y como se indica en su página, el objetivo de esta herramienta es facilitar la generación de código con seguridad de tipos C. Aunque en versiones anteriores se hiciera uso de las funciones de la API C de bajo nivel y estructuras de datos y sea un enfoque válido, su uso no se recomienda debido a que la API de bajo nivel carece de seguridad de tipos.

Como se puede ver en el manual de OMLscaffold [21] conceptualmente se definen diversas constantes como metadatos como la versión de la aplicación, la descripción… Aunque el cuerpo de la definición conceptual está en la definición de propiedades. Dónde se encuentran los parámetros del programa como el nombre, la descripción (opcional), parámetro de entrada y el tipo de dato OML. Siendo el último una abstracción de los tipos de datos de SQL de forma que no importe si la implementación real es PostgreSQL o SQLite. Posteriormente se definen los puntos de medida (MPs) que constan de una medida, o *Measurement*, que contiene diversas métricas, o *Metrics*. Dónde entre otras cosas se define que cada métrica perteneciente a una medida se mide un tipo de dato OML y cuyos valores serán reportados en una estructura del tipo MP al servidor.

2.5.3 Funcionamiento

Se comienza con la fase de inicialización y arranque de la aplicación donde se ajustan las estructuras internas y parámetros de la biblioteca OML y posteriormente se añaden las cabeceras de los metadatos de OML.

Más adelante se pasa a la fase de preparación donde se reportan las estructuras conceptuales o puntos de medida por un buffer de texto de la forma "*métrica: tipo de dato"*, que fueron indicados en el esquema procesado por el asistente OML scaffold, para pasar ya a la fase de inyección de medidas.

Ya en la fase de inyección de medidas se realizan las inyecciones según se vayan generando en el cliente al punto de recolección, o servidor OML. Si ocurriera algún error en la red o a la hora de reportar los datos se reenviarían las cabeceras y se continuaría con las inyecciones.

Acabando finalmente con un cierre de la conexión e inyección de medidas.

A continuación se ilustra con un diagrama de secuencia todos los pasos que se realizan en una aplicación general que utilice OML.

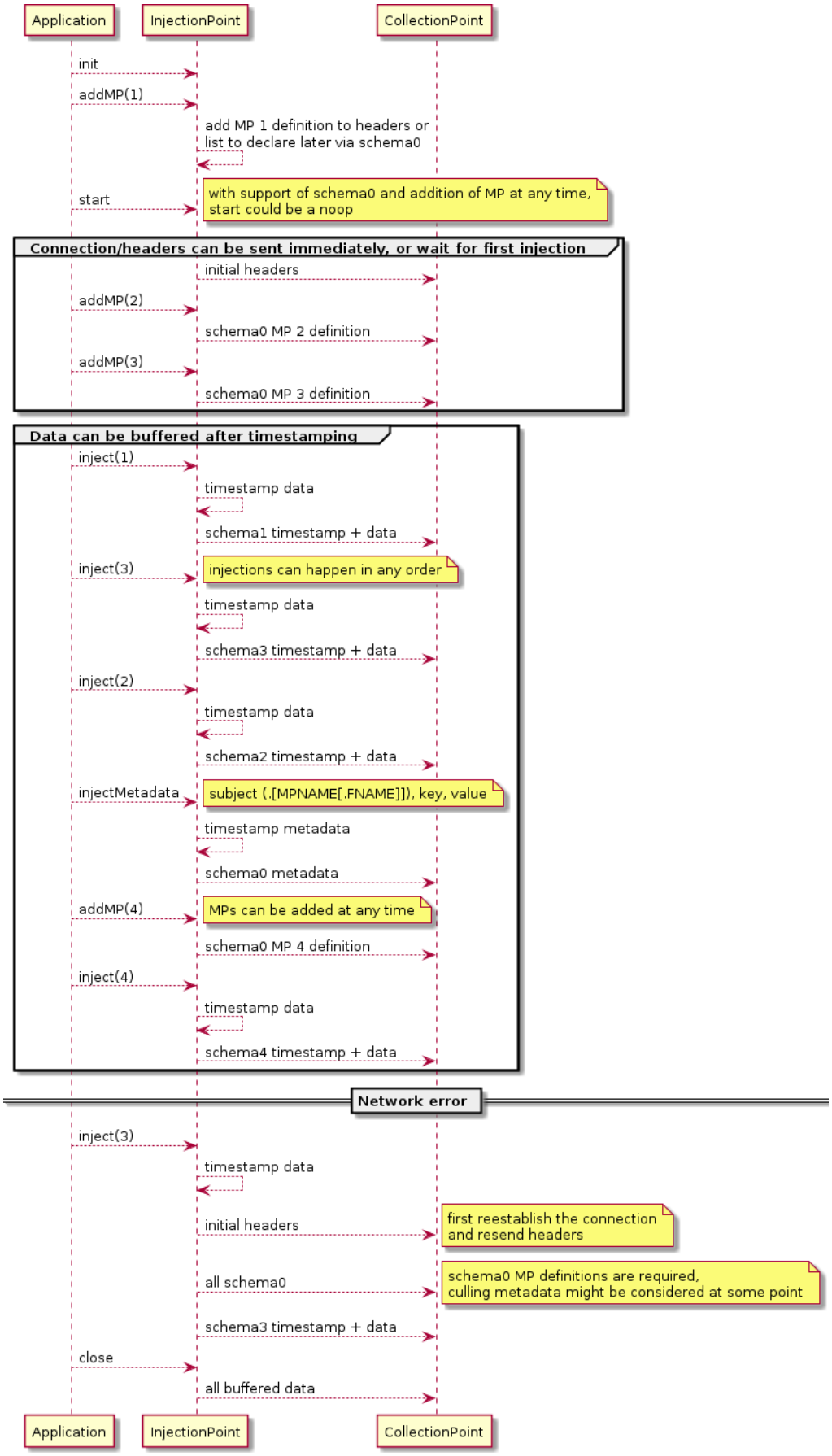


Figura 2-14: Diagrama de secuencia del funcionamiento de OML. [7]

2.6 Proyecto OpenLab

2.6.1 Introducción

OpenLab [1] es un proyecto europeo situado en el séptimo programa marco FP7 que pretende, entre otras cosas, proporcionar un conjunto testbeds para realizar experimentos en redes a gran escala. El proyecto está compuesto por varios bancos de pruebas entre los que se incluyen equipamiento de red y herramientas para obtener medidas de retardos y pérdidas de paquetes, uso de ancho de banda de enlace, seguimiento de trenes de paquetes, etc.

2.6.2 Arquitectura semántica

Dentro de este proyecto el trabajo de la UAM entre otros era el de unificar los esquemas de datos de las medidas de los testbeds [2] [15] [41]. Para ello se planteó la arquitectura SPQR que parte de un servidor común que contiene varios puntos de anclaje con diversos servidores D2R al que se pueden lanzar consultas en el lenguaje SPARQL [29] debido a la correspondencia que se realiza entre una base de datos relacional y los conceptos definidos en la ontología de MOI ETSI.

De forma que cada consulta SPARQL se difunde por todos los servidores D2R dando cada uno una respuesta con los datos pedidos por el agente que ejecutó la consulta.

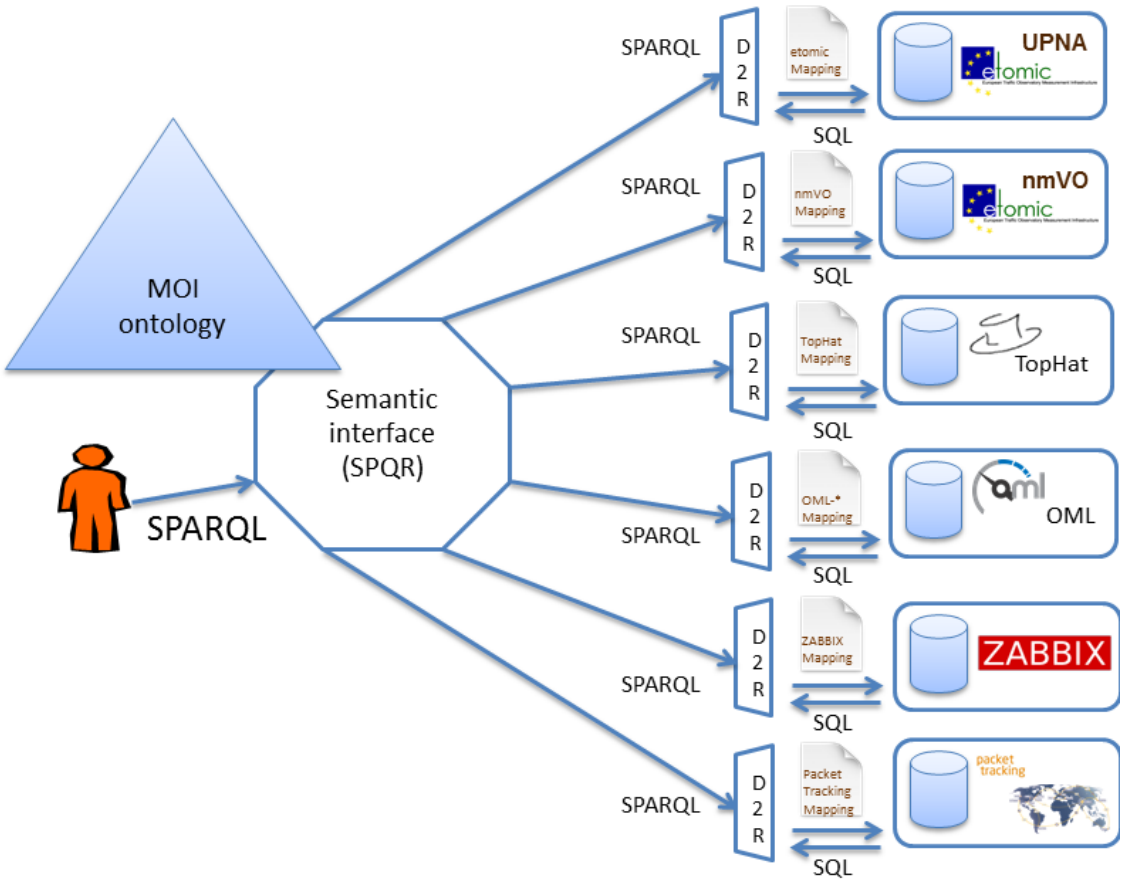


Figura 2-15: Arquitectura de SPQR.

De esta forma, se realizaron diversos mappings para adaptar los datos de los distintos experimentos:

- ETOMIC [7]: Almacena la medición de alta precisión distribuida a través de Europa por la infraestructura Observatorio de Medición de Tráfico Europeo, que es capaz de hacer medidas entre las cajas ANME especialmente diseñadas por OneLab.

- nmVO [8]: El observatorio virtual de mediciones de red es un repositorio de datos y una plataforma de compartición de datos bajo la infraestructura de ETOMIC.

- redirisMRTG [9]: RedIRIS es la red académica y de investigación española y proporciona servicios avanzados de comunicaciones a la comunidad científica y universitaria estatal.

- TopHat [10]: El proyecto TopHat, iniciado por UPMC durante el proyecto OneLab, mide activamente la topología de internet, y proporciona en vivo e históricamente medidas parar adaptar los datos de los experimentos de diferentes bases de datos.

- Zabbix [14] es un sistema de monitorización del funcionamiento de sistemas. Se instala un programa cliente en cada máquina para medir su estado y otro servidor en la máquina que colecta toda la información del resto para su análisis.

- Packet Tracking [13] es una herramienta que permite seguir el itinerario de un tren de paquetes a lo largo de diferentes nodos de una red.

2.6.3 Funcionamiento de la arquitectura SPQR

Se monta un servicio llamado SPQR [42] sobre un servidor glassfish [39] sobre el que el usuario va hacer todas las consultas. Este servicio consta de:

- Una interfaz web sobre la que realizar consultas [2].

- Una base de datos Joseki[1] [24] que actúa de caché denominada SMR ante ciertas consultas ya que las bases de datos pueden estar lejos y puede producirse bastante latencia a la hora de obtener los resultados.

- Se monta un servicio por cada endpoint que indica cómo obtener cada concepto de cada una de las bases de datos.

2.7 EndPoints Semánticos

Al haber optado por una homogenización semántica basada en la ontología ETSI MOI está claro que el endpoint debe ser semántico por lo que se ha debido hacer un análisis de los algunos endpoints semánticos de forma que se pueda intentar eliminar el overhead

[1] Servidor de publicación de contenidos RDF con acceso por HTTP GET. Joseki forma parte del marco Jenna RDF.

proporcionado por D2R al aplicar la traducción de una base de datos relacional y permitir realizar consultas en bases de datos semánticas.

La primera opción es Joseki debido a que es un modelo de base de datos ya probado de base de datos en la solución utilizada anteriormente a Semantic-OML. Fue utilizado en la solución SPQR para el caché de las consultas SPARQL según se indicó líneas atrás en la sección 2.6 y de la que se tiene conocimiento empírico de que funcione correctamente. Joseki es un proyecto que se ha movido a Fuseki, por lo que se ha probado este último con la solución TDB, que es un almacén de alto rendimiento y ligero, que utiliza la biblioteca Jena Apache / Triple Store.

La segunda alternativa a probar es la versión con licencia GPL de Virtuoso OpenLink, ya que además de dar muy buenos resultados en concurrencia de consultas, permitir ejecución paralela, adaptación a los recursos computacionales, estar escrito en C++ y ser compilado, un sistema de adaptación de caché de consultas SPARQL y muy utilizado.

La tercera alternativa fue Parliament SPARQL el cual aunque fuera un endpoint SPARQL no soportaba el estándar SPARQL/UPDATE 1.1. y tenía una comunidad escasa apoyándolo.

La cuarta y última alternativa es Open RDF Sesame [49] que aunque es de código abierto no se ha incluido en este trabajo por falta de tiempo aunque el uso de este siempre que soporte SPARQL/UPDATE 1.1 se debería poder utilizar el conector de Fuseki y en caso de necesitar un usuario y contraseña el conector de Virtuoso OpenLink.

Para el análisis de las mejores alternativas se ha tenido en cuenta cada una de las alternativas y analizando según los siguientes puntos o KPIs ordenados por orden de relevancia:

2.7.1 Licencia del producto

Todos ellos tienen una versión con licencia GPL lo que permite realizar un desarrollo sin costes en la compra de la licencia.

2.7.2 Compatibilidad con estándares

Todos soportan en alguna medida las inserciones de triplas RDF pero tanto Fuseki como Virtuoso soportan el estándar SPARQL/Update 1.1. completo.

El problema se da con Parliament que maneja un subconjunto de este estándar y aunque está en desarrollo la ampliación hay algunas características básicas como la generación de claves únicas en el servidor con la función *struuid* que no está soportado y de la cual se hablará en mayor detalle en la sección 3.3.

2.7.3 Estabilidad en el mercado

El ranking realizado por db-engines [34] que sigue un método basado en la cantidad de menciones (Google y Bing), cantidad de ofertas en el mercado laboral, cantidad de profesionales que tienen conocimientos en el sistema (linkedin), su frecuencia en discusiones técnicas, que utilizan la herramienta y relevancia en redes sociales (twitter).

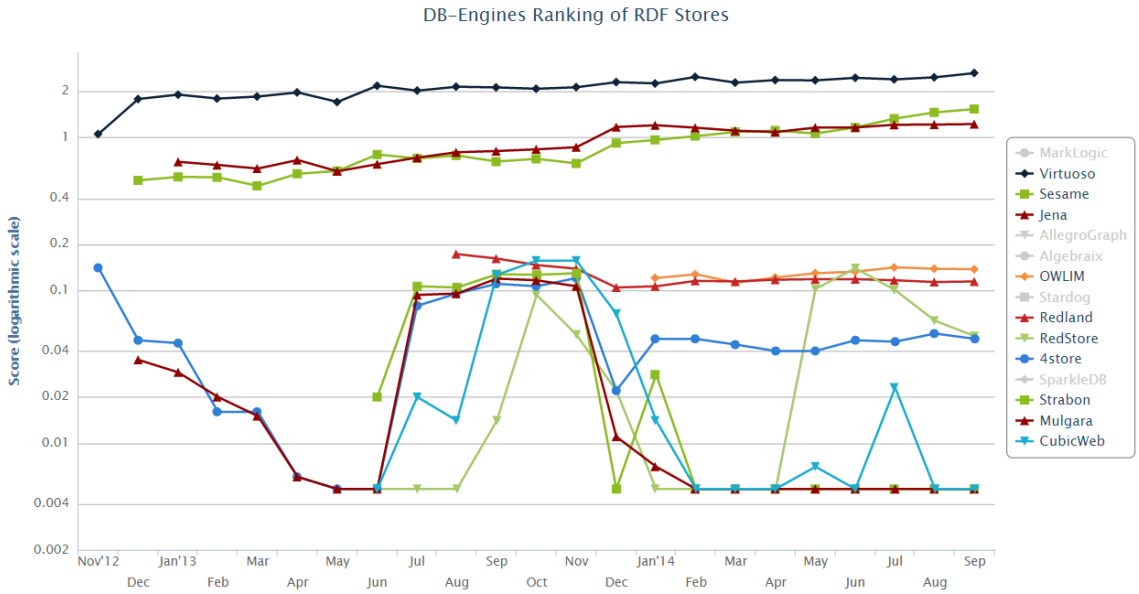


Figura 2-16: Ranking de db-engines para todos los endpoints semánticos con licencia libre. [34]

Se puede ver que algunas de las mejores a día de hoy, septiembre de 2014, con licencia open source son Virtuoso y Jena.

Para escoger el componente que gestiona el almacenamiento de RDF y consulta se ha optado por TDB [14], la recomendación actual de Jena [35], la biblioteca para web semántica de licencia Apache, escrita en Java y utilizada por un amplio marco en el mercado. TDB soporta toda la gama de APIs Jena y puede ser utilizado como almacén de alto rendimiento en una sola máquina.

2.8 Conclusiones

El análisis del estado actual del arte indica que existen diversas formas de homogenización de datos pero se ha escogido el del uso de una ontología por llevar la base de conocimiento al plano semántico donde se trabaja con conceptos. Aunque existen otras alternativas como realizar vistas de SQL o establecer un esquema común en XML se ha escogido la ontología por manejar el ámbito semántico además de haber un proceso de estandarización.

Se han analizado los protocolos más utilizados para consultas y modificación de grafos semánticos que son los que se utilizarán en el trabajo como son SPARQL, SPARQL/UPDATE 1.1. y SPARQL/UPDATE 1.1. sobre HTTP.

Se ha indicado que aunque este en proceso de estandarización aún, existe una ontología ETSI MOI que será la que se utilice como modelo ya que puede ser ampliada, mejorada con el tiempo.

Dentro del contexto OpenLab para algunos de sus bancos de pruebas se ha aplicado la alternativa de utilizar la biblioteca D2R que proporciona un endpoint semántico a partir de una base de datos relacional.

Finalmente, aunque se han visto diversos modelos para el endpoint se ha escogido analizar como alternativa a D2R, Fuseki (un servidor muy ligero) y Virtuoso (algo más pesado pero más configurable, con gran soporte en el mercado y muy utilizado).

En los próximos capítulos se introducirá la ampliación semántica de forma que se permita trabajar con las bases de datos semánticas expuestas en este capítulo como extensión de la biblioteca OML.

3 Ampliación Semántica (Semantic-OML)

3.1 Introducción

Una vez se ha mostrado en el capítulo anterior el estado del arte, en este se explica el mecanismo seguido para realizar la ampliación semántica de OML. En concreto, la que establece un esquema semántico a los puntos de medida (*Measurement Points*, MPs). Pudiendo así ocurrir que los datos reportados por esta biblioteca puedan ser almacenados en una base de datos semántica en lugar de una relacional.

En primer lugar se explica cómo se realiza la definición semántica y su validación con la ontología de ETSI MOI en la sección 3.2. A continuación, se expone el mecanismo para la inserción de la información semántica y los datos en el apartado 3.3. Finalmente, se comentan como se han realizado las aplicaciones de ejemplo para futuros desarrollos en el último apartado de esta sección.

3.2 Ampliación Semántica de OML scaffold

Para comenzar la ampliación semántica de OML es necesario añadir la definición conceptual a los puntos de medida de forma que se indique con que concepto y reglas se relaciona cada MP para ello se ha extendido el asistente que generaba el código esqueleto de la aplicación cliente OML a partir de la definición conceptual y sus MPs que se comentó en la sección 2.5.2.

Aunque al principio se barajó la posibilidad de hacer una asignación simple de medida con el concepto (Measurement) y métrica con concepto relacionado con el dato (MeasurementData) existen problemas porque no da posibilidad a añadir facetas tales como unidades, e incluso no respetar el modelo de correspondencia semántica básica explicada en la sección 2.3.

Posteriormente se pensó en que al punto de medida se le asignara un concepto del tipo Measurement y posteriormente a cada métrica del punto de medida se le asignara conceptos del tipo MeasurementData pero aquí el problema reside en no dar la flexibilidad de producir MPs con metadatos o cualquier caso especial en el que no se produzca esto y por tanto se restringe el número de problemas cubiertos. Finalmente se ha optado por la opción más flexible y es que cada métrica que estuviera dentro de un punto de medida dispusiera un concepto Measurement o de cualquier tipo pero siempre siendo el mismo de forma que todo un MP se relaciona con el mismo concepto.

De esta forma para el programa de Iperf, que identifica una conexión entre dos puntos o nodos entre los cuales la aplicación lanza paquetes para medir la capacidad, pérdidas o jitter de la conexión. Si se tiene, por ejemplo, el siguiente MP que contiene las medidas de la conexión:

Name	Type	Description
connection_id	integer	Connection identifier (socket)
local_address	string	Local network address
local_port	integer	Local port

remote_address	string	Remote network address
remote_port	integer	Remote port

Por lo que el campo connection_id la conexión tiene las siguientes relaciones:

$$MD{:}\,Iperf\ MD{:}\,hasMetricAttributes\ MD{:}\,id$$
$$MD{:}\,id\ MD{:}\,idValue\ \ll valor \gg$$
$$MD{:}\,id\ xsd{:}\,label\ "connection/connection\_id"$$

Para el campo local_address se especificaría:

$$MD{:}\,Iperf\ MD{:}\,hasMetricAttributes\ MD{:}\,SourceIP$$
$$MD{:}\,SourceIP\ MD{:}\,SourceIPValue \ll valor \gg$$
$$MD{:}\,SourceIP\ MU{:}\,defaultUnit\ MU{:}\,ipv4int$$
$$MD{:}\,SourceIP\ xsd{:}\,label\ \text{connection/local\_address}$$

Para el campo local_port se especificaría:

$$MD{:}\,Iperf\ MD{:}\,hasMetricAttributes\ MD{:}\,SourcePort$$
$$MD{:}\,SourcePort\ MD{:}\,SourcePortValue \ll valor \gg$$
$$MD{:}\,SourcePort\ xsd{:}\,label\ \text{connection/source\_port}$$

Para el campo remote_address se especificaría:

$$MD{:}\,Iperf\ MD{:}\,hasMetricAttributes\ MD{:}\,DestinationIP$$
$$MD{:}\,DestinationIP\ MD{:}\,DestinationIPValue \ll valor \gg$$
$$MD{:}\,DestinationIP\ MU{:}\,defaultUnit\ MU{:}\,ipv4int$$
$$MD{:}\,DestinationIP\ xsd{:}\,label\ \text{connection/remote\_address}$$

Para el campo remote_port se especificaría:

$$MD{:}\,Iperf\ MD{:}\,hasMetricAttributes\ MD{:}\,DestinationPort$$
$$MD{:}\,DestinationPort\ MD{:}\,DestinationPortValue \ll valor \gg$$
$$MD{:}\,DestinationPort\ xsd{:}\,label\ \text{connection/remote\_port}$$

De forma que se tendría la correspondencia entre cada uno de los conceptos y sus relaciones. Sólo falta señalar que $\ll valor \gg$ significa el valor que toma el campo. Para más detalles de la adaptación de Iperf se puede ver la sección 3.4.1.

3.2.1 Generación del código de reporte

3.2.2 Validación de conceptos

3.2.2.1 Validación con la ontología

La extensión semántica de OML scaffold realiza una validación con la ontología, siempre que se especifique el directorio donde está la definición de ésta y tener, si no lo estuviera ya instalado en la máquina, el módulo de *rdf* de Ruby [26] además de un parámetro que indique el directorio donde está la ontología de ETSI MOI. De esta forma si el usuario se equivoca al poner un concepto le devuelve una advertencia por la salida de aviso o *warning* cuando genera el código del cliente OML y también cuando ejecute el cliente OML:

```
Concept MD:PacketTrainMeasurement is not in the ETSI MOI Ontology
Concept MD:MeasurementhasMeasurementData is not in the ETSI MOI Ontology
```

De esta forma no se invade demasiado al desarrollador para que, aunque esté mal especificado, se pueda correr la aplicación, pero indicando siempre que se está realizando una definición que no está acorde con la definición de la ontología.

3.2.2.2 Algoritmo de Levensthein

Para añadir una ayuda de cara al desarrollo de la definición semántica de los MPs se realiza junto a la validación con la ontología una comparación con cada uno de los conceptos de la ontología que se correspondan al espacio de nombres reflejado en el concepto. De esta forma junto con el *warning* anterior se puede sugerir el concepto que más se parezca por distancia de Levensthein [46].

En concreto este algoritmo obtiene el mínimo número de operaciones para transformar una cadena de caracteres en otra, de forma que si el programador se equivoca al codificar el concepto poniendo, por ejemplo, MD:Iperf le devolvería MD:Iperf por ser el que más se parece con una distancia de Levensthein de 1. Pero para añadir esta mejora se debería añadir el módulo de comparación de textos de Ruby [27].

3.2.2.3 Algoritmo White Similarity

De forma parecida a la distancia de Levensthein se ha aplicado otro algoritmo que consiste en la similitud entre palabras pero en el parecido de las palabras que tiene la siguiente definición:

$Similitud(s1, s2) = \frac{2x\ |pairs(s1) \cap pairs(s2)|}{|pairs(s1)|+|pairs(s2)|}$, donde: $pairs(s) = \{t\ \epsilon\ s\}$

Por lo que por ejemplo:

$$Similitud(FRANCE, FRENCH) = \frac{2x\ |\{FR,NC\}|}{|\{FR,RA,AN,NC,CE\}|+|\{FR,RE,EN,NC,CH\}|} = \frac{2x2}{5+5} = 40\%.$$

Aunque hay un artículo con más información [47] se puede derivar al ver este algoritmo que permite obtener palabras similares a otra no solo comparando caracteres sino que elimina en parte el orden de palabras de forma que si el programador se equivoca al poner MD:SourceIP por MD:IPSource se obtendría la primera palabra con una coincidencia total.

$$Similitud(SourceIP, IPSource) = \frac{2x\ |\{SO,OU,UR,RC,CE,EI,IP\}|}{|\{SO,OU,UR,RC,CE,EI,IP\}|+|\{EI,IP,SO,OU,UR,RC,CE\}|} = 100\%.$$

En el caso de tener el concepto MD:PacketTrainMeasurement se obtiene el concepto correcto que es MD:TrainPacketMeasurement en lugar de la sugerencia de la distancia Levensthein que es MD:PacketLossMeasurement.

3.2.2.4 Resultados

En la siguiente tabla se puede ver las sugerencias proporcionadas por OML scaffold utilizando ambos algoritmos y como se puede visualizar *White Similar* (la segunda

sugerencia) obtiene mejores resultados. Teniendo en cuenta que cuando coinciden ambas sugerencias se muestra solo una.

```
WARN  Concept MD:PacketTrainMeasurement is not in the Moment Ontology.
Sugestions: "MD:PacketLossMeasurement" or "MD:TrainPacketMeasurement"

WARN  Concept MD:MeasurementhasMeasurementData is not in the Moment
Ontology. Sugestions: "MD:hasMeasurementData"

WARN  Concept MD:MeasurementhasData is not in the Moment Ontology.
Sugestions: "MD:MeasurementData"

WARN  Concept MD:IPDirectionDestination is not in the Moment Ontology.
Sugestions: "MD:DestinationIP"

WARN  Concept MD:Trace is not in the Moment Ontology. Sugestions:
"MD:Type" or "MD:Traceroute"
```

Figura 3-1: Correcciones del esquema semántico definido para OML scaffold.

3.3 Inserción de información semántica

Una vez se tiene la definición semántica del punto de medida ya solo resta mantenerla en todo el recorrido de OML, es decir, recuperar esta información de la estructura desde el cliente OML, enviarla por el buffer de datos ya sea binario o de texto al servidor OML al enviar la cabecera del MP y finalmente escribir los resultados en cada inserción.

3.3.1 Obtención de los conceptos semánticos en el cliente OML

Teniendo en cuenta que OML scaffold genera el esqueleto de la aplicación cliente para la obtención de los conceptos únicamente se debe recoger los conceptos y para ellos se añade a la estructura ya existente que especifica el esquema relacional los conceptos con los que se relaciona cada campo o métrica y las relaciones entre estos.

Para ello, sobre la estructura del punto de medida que almacenaba el nombre de esta y sus métricas o campos se ha añadido el concepto del tipo Measurement, el concepto del tipo MeasurementData y finalmente las facetas de la métrica como la unidad en que se mide.

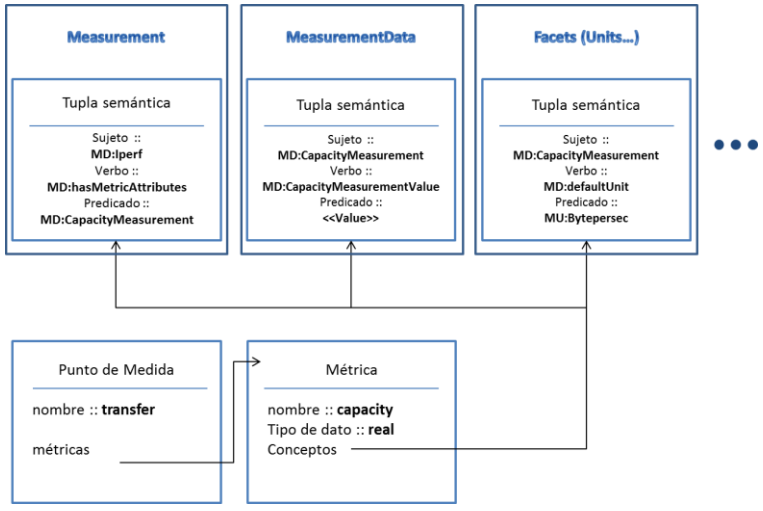


Figura 3-2: Esquema semántico de ejemplo en una aplicación cliente Semantic-OML.

Es importante indicar que aunque es aconsejable que las métricas de una misma medida estén relacionadas con un mismo concepto del tipo Measurement se permiten establecer distintos conceptos.

3.3.2 Envío de esquema conceptual semántico al servidor OML

El reporte de medidas de OML tiene dos partes bien diferenciadas en el reporte entre la aplicación cliente y el servidor que ya se indicaron en la sección 2.5 y son el envío de las cabeceras de los puntos de medida que contienen el esquema de los datos y la segunda parte que se corresponde con los datos de las medidas.

Para mantener la secuencia del flujo de datos se realiza el envío del esquema conceptual que mantiene una estructura similar a definida en el apartado anterior junto con las cabeceras especificando además del nombre y tipo de dato de la métrica los conceptos relacionados.

No obstante, existe la posibilidad en OML de utilizar tanto un buffer de texto como uno binario para el envío de datos. En el caso del esquema relacional como ya se indicó anteriormente la cabecera del esquema de datos se envía siempre el nombre de punto de medida, métricas y sus tipos de datos en un buffer de texto.

Pese a que añadir el esquema semántico a las cabeceras aumenta el tamaño de estas. Si se tiene en cuenta que convertir este buffer a uno binario supone la codificación binaria de cada concepto de la ontología lo que obliga a una menor flexibilidad propiciando a actualizar a nivel de código de Semantic-OML. De forma que para cada modificación se obtiene una pequeña ganancia en la disminución de los datos transmitidos. Esto supone una pequeña fracción en el proceso de reporte de medidas en el caso de que dedique más tiempo a la obtención y envío de datos, que suele ser el caso más común, por lo que se ha decidido mantener el buffer de texto.

3.3.3 Generación de las sentencias de inserción (SPARQL 1.1)

Tras recibir las cabeceras OML con *backend* relacional establece el esquema de tablas y genera las sentencias de inserción y de creación de tablas. De forma similar se ha añadido que Semantic-OML genere las sentencias de inserción con la diferencia de que con la base de datos semántica no se genera una estructura debido a que por naturaleza consta de triplas y no existe una estructura entre estas.

No obstante, aunque no exista una estructura clásica sí que existe un patrón en el reporte de los datos y sirviéndose de este se realiza la generación de las sentencias de inserción.

Cómo ya se indicó en el apartado 2.4 el esquema conceptual de los datos parte de un concepto del tipo Measurement el cual está relacionado con conceptos MeasurementData y finalmente se indican las facetas de estos datos como las unidades en que fueron medidos o el tipo de dato. De esta forma se tiene en cuenta que un punto de medida puede estar relacionado con un concepto de tipo Measurement relacionando cada una de sus métricas con un concepto de tipo MeasurementData las cuales tienen un valor determinado y son medidas en una unidad determinada.

Teniendo en cuenta que se tiene el siguiente esquema conceptual:

Medida (Measurement)

- Nombre
- Métricas:
  - Métrica 1
    - Nombre de la métrica
    - Tipo de dato de la métrica [entero, real, string…]
    - Esquema semántico de la métrica
      - Measurement1 – Relación M-MD1 – MeasurementData1.
      - MeasurementData1 – Relación MD-Valor1 – Valor1.
      - MeasurementData1 – Relación MD-Faceta1 – Faceta1.
      - MeasurementData1 – Relación MD-Faceta2 – Faceta2.
      - …
  - […]
  - Métrica N
    - Nombre de la métrica
    - Tipo de dato de la métrica [entero, real, string…]
    - Esquema semántico de la métrica
      - Measurement1 – Relación M-MD2 – MeasurementData2.
      - MeasurementData2 – Relación MD-Valor2 – Valor2.
      - MeasurementData2 – Relación MD-Faceta3 – Faceta3.
      - …

Siguiendo el estándar SPARQL / Update 1.1 definido por W3C [9] se genera de una métrica general con la sintaxis:

INSERT DATA

{

M1 a Measurement1**;**

Label *"Nombre de la medida"^^xsd:string* **;**

Relación M-MD1 *MD1* **;**

Relación M-MD2 *MD2* **;**

… **.**

MD1 a MeasurementData1 **;**

Label "Nombre de la métrica 1*"^^xsd:string* **;**

Relación MD1-Valor1 "Valor1"^^xsd:"Tipo de dato de la métrica" **;**

Relación MD1-Faceta1 Faceta1 **;**

```
                Relación MD1-Faceta2 Faceta ;

                … .


        MD2 a MeasurementData2;

                Label “Nombre de la métrica 2” ^^xsd:string;

                Relación MD2-Valor2 Valor2^^xsd:”Tipo de dato de la métrica”;

                Relación MD2-Faceta3 Faceta3 ;

                … .
}
```

Dónde M{*i*}, MD{*i*}, Valor{*i*} y Faceta{*i*} son instancias conceptuales e {*i*} son identificadores únicos de cada una de las instancias, más formalmente:

$$Instancia\ Conceptual = \langle I \epsilon \{M, MD, Valor, Faceta\}, \forall i, j / i \neq j | I_i \rangle$$

Es importante indicar que se utiliza el concepto de grafo como contenedor de almacén de triplas RDF que puede ser definido como un par de base de datos semántica con una IRI asociada definido en el estándar SPARQL 1.1. [29]. Definiendo que hay un grafo por cada dominio experimental siendo identificado por su IRI compuesto por el host y puerto del servidor de Semantic-OML y dominio indicado en las aplicaciones clientes Semantic-OML. Permitiendo así diferenciar los conjuntos de datos diferenciando el dominio (o grafo semántico) o uniendo dominios (o grafos semánticos).

Una vez que se han generado las plantillas de inserción para cada reporte correspondiente a un punto de medida se asignan los valores de cada métrica a cada valor correspondiente generando las sentencias de forma más rápida y pudiendo enviar al *endpoint* semántico.

3.3.4 Envío de las sentencias de inserción (SPARQL 1.1)

El estándar SPARQL/UPDATE 1.1 [9] establece el protocolo para realización de inserciones SPARQL/Update sobre el protocolo de comunicación HTTP siendo ésta la solución más general y adoptada debido a que está establecido como un estándar permite una gran cantidad de *endpoints* semánticos posibles aunque en este proyecto se ha operado con dos, Fuseki+TDB y Virtuoso OpenLink+TDB.

Respecto a la unicidad de las inserciones un problema que ha llevado bastante tiempo de investigación se ha atajado con la función definida en el propio estándar SPARQL 1.1. STRUUID que genera de forma atómica un identificador único para cada elemento de la base de datos, algo que ya se tenía con D2R con la función urify. El problema reside en que el estándar no define un mecanismo de transacciones lo que obligaba a no poder utilizar bloqueos en la generación de identificadores, ni tampoco se podían generar estos en la aplicación que reporta los datos debido a que el cliente no puede obtener un identificador único, por ello debe utilizarse la función indicada en el servidor.

Para el envío de las sentencias al utilizar el protocolo HTTP se ha añadido la necesidad de manejar la librería cURL que permite conectar con cualquier host, añadir cabeceras HTTP y el *payload* necesario de forma sencilla y sin un alto coste computacional.

Este sistema tiene una particularidad respecto al anterior y es el coste de mantener una conexión HTTP y la latencia por transacción debida al uso del protocolo SPARQL/Update 1.1 que provocará que a tasas demasiado altas de reporte de datos y un sistema lento en *backend* pueda llegar a producir pérdidas con mayor probabilidad en las medidas que pueden arreglarse de manera temporal con un buffer más grande si el tiempo de cómputo y procesado por inserción supera a la tasa de llegadas el sistema está condenado en este caso a llegar al estado de saturación.

3.4 Añadir la información semántica a algunos de los ejemplos de Semantic-OML

Junto con el Framework OML viene un paquete con varios ejemplos de la utilización de la librería. En este apartado se analizarán tres ejemplos y se detallará su adaptación para poder detallar empíricamente el funcionamiento de la extensión semántica.

3.4.1 Iperf

3.4.1.1 Definición y adaptación de OML

Existen multitud de herramientas multiplataforma para la evaluación de rendimientos en las comunicaciones redes de computadores y posterior optimización de los parámetros. Una de ellas es Iperf [36], con la que se puede medir el ancho de banda máximo TCP, lo que permite la afinación de diversos parámetros y características para UDP. Iperf informa acerca del ancho de banda, jitter y pérdida de paquetes.

Más en detalle si se configura para una conexión TCP se obtiene el ancho de banda, el tamaño de MSS/MTU, soporta definir un tamaño de pantalla utilizando sockets con buffers, puede realizar diversas conexiones simultáneas gracias a su opción multi-hilo. En el caso de UDP, se puede generar flujos basados en el ancho de banda para medir la pérdida de paquetes, retardo y jitter. Además en ambos casos se puede especificar en sus opciones la unidad en que recolecta medidas a ser: K (kilo-), M (mega-)…, especificar el tiempo de transferencia de datos.

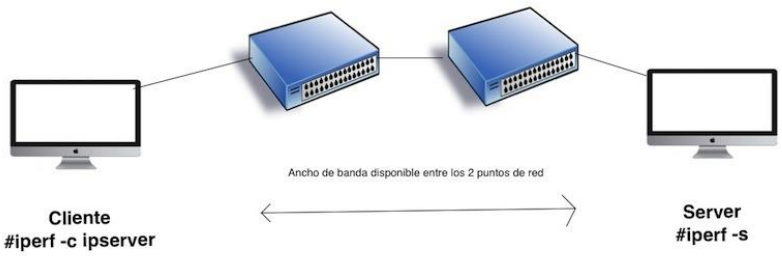


Figura 3-3: Esquema semántico de ejemplo en una aplicación cliente Semantic-OML.

Junto con las aplicaciones de ejemplo de OML viene la herramienta de medición de conexiones Iperf. En la que en primera instancia se definió un conjunto de puntos de

medida o MPs. Entre ellos se definen la aplicación, que identifica el proceso que se ha ejecutado y sus parámetros; las opciones, que proporciona un detalle de las opciones utilizadas en el proceso; la conexión, que identifica la conexión entre el cliente y el servidor OML que intervienen en la medición; la transferencia, que almacena el intervalo entre envío de datos y la longitud de estos; las pérdidas, que contiene la información acerca de las pérdidas entre intervalos de la medición; el jitter, o varianza en los retardos entre intervalos de las medidas; y finalmente el punto de medida de paquetes, que contiene el tamaño de los paquetes y el retardo en la transferencia de estos.

3.4.1.2 Adaptación de Semantic-OML

Para la adaptación semántica se han asociado los conceptos y las métricas de la siguiente manera:

- Punto de medida “application”:

```
id
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
   ⇨ MD:id → MD:MeasurementDataValue → <<value>>

version
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:ToolVersion
   ⇨ MD:ToolVersion → MD:ToolVersionValue → <<value>>

cmdline
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:Arguments
   ⇨ MD:Arguments → MD:ArgumentsValue → <<value>>
```

- Punto de medida “settings”:

```
id
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
   ⇨ MD:id → MD:MeasurementDataValue → <<value>>

bind_address
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:BindIP
   ⇨ MD:BindIP → MD:BindIPValue → <<value>>
   ⇨ MD:BindIP → MD:defaultUnit → MU:ipv4dotted

multicast
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:isListeningMulticast
   ⇨ MD:isListeningMulticast → MD:isListeningMulticastValue → <<value>>

Multicast_ttl
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:FinalTtlMeasurement
   ⇨ MD:FinalTtlMeasurement → MD:FinalTtlMeasurementValue → <<value>>

transport_protocol
   ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TransportProtocol
   ⇨ MGC:TransportProtocol → MGC:protocolNumber → <<value>>

window_size
   ⇨ MD:Iperf → MD:hasMetricAttributes → MD:Arguments
```

```
    ⇨ MD:Arguments → MD:ArgumentsValue → <<value>>

buffer_size
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:Arguments
    ⇨ MD:Arguments → MD:ArgumentsValue → <<value>>
```

- Punto de medida “connection”:

```
id
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
    ⇨ MD:id → MD:MeasurementDataValue → <<value>>

local_address
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:SourceIP
    ⇨ MD:SourceIP → MD:SourceIPValue → <<value>>
    ⇨ MD:SourceIP → MD:defaultUnit → MU:ipv4dotted

local_port
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:SourcePort
    ⇨ MD:SourcePort → MD:SourcePortValue → <<value>>

remote_address
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:DestinationIP
    ⇨ MD:DestinationIP → MD:DestinationIPValue → <<value>>
    ⇨ MD:SourceIP → MD:defaultUnit → MU:ipv4dotted

remote_port
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:DestinationPort
    ⇨ MD:DestinationPort → MD:DestinationPortValue → <<value>>
```

- Punto de medida “transfer”:

```
id
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
    ⇨ MD:Iperf → MD:MeasurementDataValue → <<value>>

begin_interval
    ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:startTime → <<value>>
    ⇨ MGC:TimeStamp → MD:defaultUnit → MU:second

end_interval
    ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:endTime → <<value>>
    ⇨ MGC:TimeStamp → MD:defaultUnit → MU:second

capacity
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:CapacityMeasurement
    ⇨ MD:CapacityMeasurement → MD:CapacityMeasurementValue → <<value>>
    ⇨ MD:CapacityMeasurement → MD:defaultUnit → MU:Bytepersec
```

- Punto de medida “losses”:

```
id
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
    ⇨ MD:Iperf → MD:MeasurementDataValue → <<value>>

begin_interval
    ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:startTime → <<value>>
    ⇨ MGC:TimeStamp → MD:defaultUnit → MU:second

end_interval
    ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:endTime → <<value>>
    ⇨ MGC:TimeStamp → MD:defaultUnit → MU:second

total_datagrams
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:PacketsCount
    ⇨ MD:PacketsCount → MD:PacketsCountValue → <<value>>

lost_datagrams
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:ErrorsRate
    ⇨ MD:ErrorsRate → MD:ErrorsRateValue → <<value>>
```

- Punto de medida “jitter”:

```
id
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
    ⇨ MD:Iperf → MD:MeasurementDataValue → <<value>>

begin_interval
    ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:startTime → <<value>>
    ⇨ MGC:TimeStamp → MD:defaultUnit → MU:second

end_interval
    ⇨ MD:Iperf → MD:hasMetricAttributes → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:endTime → <<value>>
    ⇨ MGC:TimeStamp → MD:defaultUnit → MU:second

jitter
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:DelayVariationMeasurement
    ⇨ MD:DelayVariationMeasurement → MD:DelayVariationMeasurementValue →
      <<value>>
    ⇨ MD:CapacityMeasurement → MD:defaultUnit → MU:millisecond
```

- Punto de medida “packets”:

```
id
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:id
    ⇨ MD:Iperf → MD:MeasurementDataValue → <<value>>

packet_id
```

```
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:PacketIdentifier
    ⇨ MD:PacketIdentifier → MD:PacketIdentifierValue → <<value>>

packet_size
    ⇨ MD:Iperf → MD:hasMetricAttributes → MD:PacketSize
    ⇨ MD:PacketSize → MD:PacketSizeValue → <<value>>
    ⇨ MD:PacketSize → MD:defaultUnit → MU:Byte

packet_time_s
    ⇨ MD:Iperf → MD:hasMeasurementData → MD:OneWayDelayMeasurement
    ⇨ MD:OneWayDelayMeasurement → MD:OneWayDelayMeasurementValue →
      <<value>>
    ⇨ MD:OneWayDelayMeasurement → MD:defaultUnit → MU:second

packet_time_us
    ⇨ MD:Iperf → MD:hasMeasurementData → MD:OneWayDelayMeasurement
    ⇨ MD:OneWayDelayMeasurement → MD:OneWayDelayMeasurementValue →
      <<value>>
    ⇨ MD:OneWayDelayMeasurement → MD:defaultUnit → MU:second

packet_sent_time_s
    ⇨ MD:Iperf → MD:hasMeasurementData → MD:OneWayDelayMeasurement
    ⇨ MD:OneWayDelayMeasurement → MD:OneWayDelayMeasurementValue →
      <<value>>
    ⇨ MD:OneWayDelayMeasurement → MD:defaultUnit → MU:second

packet_sent_time_us
    ⇨ MD:Iperf → MD:hasMeasurementData → MD:OneWayDelayMeasurement
    ⇨ MD:OneWayDelayMeasurement → MD:OneWayDelayMeasurementValue →
      <<value>>
    ⇨ MD:OneWayDelayMeasurement → MD:defaultUnit → MU:second
```

3.4.2 Trace

3.4.2.1 Definición y adaptación OML

Trace es una herramienta que utiliza la biblioteca libtrace [37] que captura paquetes obteniendo la información de estos acerca de los parámetros y cabeceras de IP, TCP y UDP. Además, permite leer ficheros de trazas con el formato PCAP [38]. Junto con Iperf permite una mejor interpretación de la calidad de la red permitiendo realizar, además, las cabeceras de los paquetes.

La adaptación consta de diferentes puntos de medida o MPs, entre ellos, la información de cabeceras de flujos IP como el offset del fragmento IP, Time to Live, protocolo, dirección origen, destino, el tamaño del paquete y el timestamp del momento de la medida; la información de flujos TCP como puerto origen y destino, la secuencia, el tamaño de ventana, el tamaño del paquete y el timestamp de cuando se realizó la medida; la información de flujos UDP como los puertos origen y destino, tamaño del paquete y timestamp de la medida.

3.4.2.2 Adaptación de Semantic-OML

En este caso para adaptar semánticamente la aplicación se ha añadido las siguientes relaciones para los siguientes puntos de medida:

- Punto de medida “ip”:

pktid
- MD:Trace → MD:hasMeasurementData → MD:PacketIdentifier
- MD:PacketIdentifier → MD:PacketIdentifierValue → **<<value>>**

ip_id
- MD:Trace → MD:hasMeasurementData → MD:IpIdentifier
- MD:IpIdentifier → MD:hasMeasurementValue → **<<value>>**

ip_ttl
- MD:Trace → MD: hasMeasurementData → MD:FinalTtlMeasurement
- MD:FinalTtlMeasurement → MD:SimpleMeasurementValue → **<<value>>**

ip_proto
- MD:Trace → MD:hasMeasurementData → MGC:TransportProtocol
- MGC:TransportProtocol → MGC:protocolNumber → **<<value>>**

ip_src
- MD:Trace → MD:hasMeasurementData → MD:SourceIP
- MD:SourceIP → MD:SourceIPValue → **<<value>>**
- MD:SourceIP → MD:defaultUnit → MU:ipv4dotted

ip_dst
- MD:Trace → MD:hasMeasurementData → MD:DestinationIP
- MD:DestinationIP → MD:DestinationIPValue → **<<value>>**
- MD:DestinationIP → MD:defaultUnit → MU:ipv4dotted

ip_sizeofpacket
- MD:Trace → MD:hasMeasurementData → MD:PacketSize
- MD:PacketSize → MD:PacketSizeValue → **<<value>>**
- MD:PacketSize → MD:defaultUnit → MU:Byte

ip_ts
- MD:Trace → MD:hasMeasurementData → MGC:TimeStamp
- MGC:TimeStamp → MGC:timestamp → **<<value>>**

- Punto de medida “tcp”:

pktid
- MD:Trace → MD:hasMeasurementData → MD:PacketIdentifier
- MD:PacketIdentifier → MD:PacketIdentifierValue → **<<value>>**

tcp_source
- MD:Trace → MD:hasMeasurementData → MD:SourcePort
- MD:SourcePort → MD:SourcePortValue → **<<value>>**
- MD:SourcePort → MD:defaultUnit → MU:ipv4dotted

tcp_dest
- MD:Trace → MD:hasMeasurementData → MD:DestinationPort
- MD:DestinationPort → MD:DestinationPortValue → **<<value>>**

tcp_seq
- MD:Trace → MD:hasMeasurementData → MD:SequenceIndex
- MD:SequenceIndex → MD:SequenceIndexValue → **<<value>>**

```
tcp_packet_size
    ⇨ MD:Trace → MD:hasMeasurementData → MD:PacketSize
    ⇨ MD:PacketSize → MD:PacketSizeValue → <<value>>

tcp_ts
    ⇨ MD:Trace → MD:hasMeasurementData → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:timestamp → <<value>>
```

- Punto de medida “udp”:

```
pktid
    ⇨ MD:Trace → MD:hasMeasurementData → MD:PacketIdentifier
    ⇨ MD:PacketIdentifier → MD:PacketIdentifierValue → <<value>>

udp_source
    ⇨ MD:Trace → MD:hasMeasurementData → MD:SourcePort
    ⇨ MD:SourcePort → MD:SourcePortValue → <<value>>

udp_dest
    ⇨ MD:Trace → MD:hasMeasurementData → MD:DestinationPort
    ⇨ MD:DestinationPort → MD:DestinationPortValue → <<value>>
    ⇨ MD:DestinationPort → MD:defaultUnit → MU:ipv4dotted

udp_len
    ⇨ MD:Trace → MD:hasMeasurementData → MD:PacketSize
    ⇨ MD:PacketSize → MD:PacketSizeValue → <<value>>
    ⇨ MD:PacketSize → MD:defaultUnit → MU:Byte

ip_ts
    ⇨ MD:Trace → MD:hasMeasurementData → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:timestamp → <<value>>
```

3.4.3 Otg2/Otr2

3.4.3.1 Definición

La aplicación OTG es un generador de tráfico UDP y OTR es el receptor de los flujos generados por el anterior y ambos reportan información acerca de los tiempos de salida y de llegada por paquete y su tamaño.

Más en detalle el generador del tráfico UDP almacena por cada paquete su identificador, el tiempo en que se manda, su tamaño y la dirección IP y puerto del destino; y de forma análoga la recepción almacena el identificador, el tamaño, la dirección IP y puerto del origen del flujo.

3.4.3.2 Adaptación de Semantic-OML

Para la adaptación semántica se han asociado los conceptos y las métricas de la siguiente manera:

- Punto de medida “udp_in” (OTR o receptor):

```
ts
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:timestamp → <<value>>

flow_id
    ⇨ MD:TrainPacketMeasurement → MD:hasMetricAttributes →
      MD:ToolVersion
    ⇨ MD:ToolVersion → MD:ToolVersionValue → <<value>>

seq_no
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData →
      MD:SequenceIndex
    ⇨ MD:SequenceIndex → MD:SequenceIndexValue → <<value>>

pkt_length
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData → MD:PacketSize
    ⇨ MD:PacketSize → MD:PacketSizeValue → <<value>>
    ⇨ MD:PacketSize → MD:defaultUnit → MU:Byte

dst_host
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData → MD:SourceIP
    ⇨ MD:SourceIP → MD:SourceIPValue → <<value>>
    ⇨ MD:SourceIP → MD:defaultUnit → MU:ipv4dotted

dst_port
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData → MD:SourcePort
    ⇨ MD:SourcePort → MD:SourcePortValue → <<value>>
```

- Punto de medida “OTR”:

```
ts
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData → MGC:TimeStamp
    ⇨ MGC:TimeStamp → MGC:timestamp → <<value>>

flow_id
    ⇨ MD:TrainPacketMeasurement → MD:hasMetricAttributes →
      MD:ToolVersion
    ⇨ MD:ToolVersion → MD:ToolVersionValue → <<value>>

seq_no
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData →
      MD:SequenceIndex
    ⇨ MD:SequenceIndex → MD:SequenceIndexValue → <<value>>

pkt_length
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData → MD:PacketSize
    ⇨ MD:PacketSize → MD:PacketSizeValue → <<value>>
    ⇨ MD:PacketSize → MD:defaultUnit → MU:Byte

dst_host
    ⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData →
      MD:DestinationIP
    ⇨ MD:DestinationIP → MD:DestinationIPValue → <<value>>
    ⇨ MD:DestinationIP → MD:defaultUnit → MU:ipv4dotted

dst_port
```

```
⇨ MD:TrainPacketMeasurement → MD:hasMeasurementData →
  MD:DestinationPort
⇨ MD:DestinationPort → MD:DestinationPortValue → <<value>>
```

3.5 Conclusiones

En resumen, se define una estructura semántica que se pasa junto con la ontología a OML scaffold para obtener el esqueleto de la aplicación cliente de Semantic-OML, con esta y el Framework Semantic-OML se compila la aplicación que realiza los reportes al servidor Semantic-OML el cual recolecta todos los datos de cada uno de los clientes y los difunde al correspondiente *endpoint* y grafo semántico utilizando el estándar SPARQL/UPDATE 1.1 sobre HTTP. Esto se puede ver esquematizado en la siguiente figura.

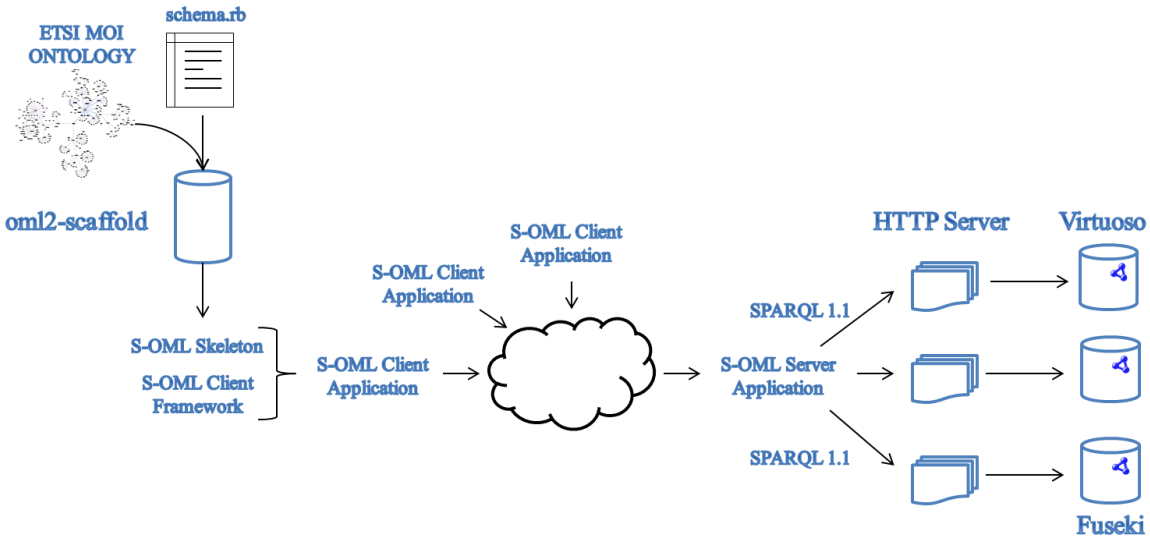


Figura 3-4: Esquema gráfico de la arquitectura de Semantic-OML.

Además se ha proporcionado un conjunto de ejemplos con los que probar este esquema y así poder validar la solución, algo que se realizará en el siguiente capítulo.

4 Validación de la solución

4.1 Introducción

En los capítulos anteriores se ha analizado el estado del arte y el trabajo realizado para poder homogenizar los diferentes bancos de datos, como ha sido la arquitectura SPQR que utiliza la ontología definida por ETSI MOI. Tras esta solución se ha planteado una ampliación semántica en la que definiendo las relaciones conceptuales definidas en la ontología se consigue almacenar las mediciones realizadas en bases de datos semánticas.

En esta sección se analizará en diferentes apartados si la solución aportada cubre las necesidades funcionales básicas, apartado 4.2. Posteriormente se analizaran las ventajas e inconvenientes del uso de las dos bases de datos semánticas escogidas, Virtuoso OpenLink y Fuseki en el apartado 4.3 y finalmente se hará una comparativa con la solución anterior a la extensión de la biblioteca en el apartado 4.4.

4.2 Validación de la solución

Para validar la solución se ha realizado una comprobación en dos partes. La primera en la que se analizará si las medidas son almacenadas satisfactoriamente en las bases de datos semánticas y la segunda se analizará si las consultas a las bases de datos semánticas devuelven los mismos resultados que la solución utilizando D2R y para ello se han utilizado los ejemplos que definidos en el apartado 3.4.

4.2.1 Validación de almacenamiento

En esta comprobación se ha realizado diversas consultas a los *endpoints* semánticos comprobando que las medidas obtenidas son almacenadas correctamente para ello se han analizado diferentes escenarios en los que se han tenido en cuenta cada aplicación de ejemplo provista.

En el caso de Iperf se han ejecutado las consultas Q1, que obtiene las direcciones IP implicadas, Q2, que obtiene además de lo anterior el ancho de banda máximo, mínimo y medio por conexión, y Q3, que obtiene además de lo anterior el ancho de banda máximo, mínimo y medio por conexión. Todas ellas, disponibles en el anexo A, han sido ejecutadas en dos casos diferentes, uno en el que se encuentra el servidor y cliente Iperf en la misma máquina y el segundo el servidor Iperf está en el campus de la UAM y el cliente en un ordenador en San Sebastián de los Reyes. Teniendo como resultados los mostrados por las siguientes tablas, se cumple la validación al almacenarse correctamente los datos.

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd
">
 <head>
  <variable name="SourceIPValue"/>
  <variable name="DstIPValue"/>
 </head>
 <results>
  <result>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">127.0.0.1</literal></
```

```
binding>
   <binding name="DstIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">127.0.0.1</literal></
binding>
  </result>
  <result>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">192.168.1.13</literal
></binding>
   <binding name="DstIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">150.244.56.136</liter
al></binding>
  </result>
</results>
</sparql>
```

Figura 4-1: Resultado de la consulta 1 para el caso 2, servidor en la UAM, cliente en San Sebastián de los Reyes.

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd
">
 <head>
  <variable name="SourceIPValue"/>
  <variable name="DstIPValue"/>
  <variable name="MinCapacityValue"/>
  <variable name="AvgCapacityValue"/>
  <variable name="MaxCapacityValue"/>
 </head>
 <results>
  <result>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">192.168.1.13</literal
></binding>
   <binding name="DstIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">127.0.0.1</literal></
binding>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#double">1611120</literal></bi
nding>
   <binding name="DstIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#double">3154446.05</literal><
/binding>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#double">4301220</literal></bi
nding>
  </result>
 </results>
</sparql>
```

Figura 4-2: Resultado de la consulta 2 para el caso 2, servidor en la UAM, cliente en San Sebastián de los Reyes.

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd
">
```

```
 <head>
  <variable name="SourceIPValue"/>
  <variable name="DstIPValue"/>
  <variable name="MinStartTime"/>
  <variable name="MaxEndTime"/>
  <variable name="MinCapacityValue"/>
  <variable name="AvgCapacityValue"/>
  <variable name="MaxCapacityValue"/>
 </head>
 <results>
  <result>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">192.168.1.13</literal
></binding>
   <binding name="DstIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">127.0.0.1</literal></
binding>
   <binding name="MinStartTime"><literal
datatype="http://www.w3.org/2001/XMLSchema#decimal">0</literal></binding
>
   <binding name="MaxEndTime"><literal
datatype="http://www.w3.org/2001/XMLSchema#decimal">600</literal></bindi
ng>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#double">1611120</literal></bi
nding>
   <binding name="DstIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#double">3154446.05</literal><
/binding>
   <binding name="SourceIPValue"><literal
datatype="http://www.w3.org/2001/XMLSchema#double">4301220</literal></bi
nding>
  </result>
 </results>
</sparql>
```

Figura 4-3: Resultado de la consulta 3 para el caso 2, servidor en la UAM, cliente en San Sebastián de los Reyes.

En el caso de Trace se ha ejecutado la consulta Q4, adjuntada en el mismo anexo que las anteriores, con la que se obtiene la información acerca del tráfico TCP/IP. En este caso se ha capturado el tráfico de una conexión HTTPS, a continuación se puede visualizar algunos datos de las cabeceras IP y TCP de dos paquetes de dicho flujo.

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd
">
 <head>
  <variable name="type"/>
  <variable name="IPidVal"/>
  <variable name="IPttlVal"/>
  <variable name="IPtsVal"/>
  <variable name="UPPtsVal"/>
  <variable name="IPproVal"/>
  <variable name="IPsrcVal"/>
  <variable name="UPPsrcPortVal"/>
  <variable name="IPdstVal"/>
  <variable name="UPPdstPortVal"/>
```

```
  <variable name="IPSizeVal"/>
  <variable name="UPPsizeVal"/>
 </head>
 <results distinct="false" ordered="true">
  <result>
   <binding name="type"><literal>trace_tcp</literal></binding>
   <binding name="IPidVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">32936</literal></bin
ding>
   <binding name="IPttlVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">64</literal></bindin
g>
   <binding name="IPtsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#decimal">0.886223</literal></
binding>
   <binding name="UPPtsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#decimal">0.886223</literal></
binding>
   <binding name="IPproVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">6</literal></binding
>
   <binding name="IPsrcVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">192.168.1.12</literal
></binding>
   <binding name="UPPsrcPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">58949</literal></bin
ding>
   <binding name="IPdstVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">74.125.230.59</litera
l></binding>
   <binding name="UPPdstPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">443</literal></bindi
ng>
   <binding name="IPSizeVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">66</literal></bindin
g>
   <binding name="UPPsizeVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">66</literal></bindin
g>
  </result>
  <result>
   <binding name="type"><literal>trace_tcp</literal></binding>
   <binding name="IPidVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">42120</literal></bin
ding>
   <binding name="IPttlVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">50</literal></bindin
g>
   <binding name="IPtsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#decimal">6.52347</literal></b
inding>
   <binding name="UPPtsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#decimal">6.52347</literal></b
inding>
   <binding name="IPproVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">6</literal></binding
>
   <binding name="IPsrcVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">173.194.6.178</litera
l></binding>
   <binding name="UPPsrcPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">443</literal></bindi
```

```
ng>
   <binding name="IPdstVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">192.168.1.12</literal
></binding>
   <binding name="UPPdstPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">55788</literal></bin
ding>
   <binding name="IPSizeVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">100</literal></bindi
ng>
   <binding name="UPPsizeVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">100</literal></bindi
ng>
  </result>
 </results>
</sparql>
```

Figura 4-4: Resultado cortado a dos paquetes transmitidos por dos flujos distintos HTTPS (protocolo 443).

Finalmente para la aplicación otg2/otr2 que genera y recoge tráfico se ha ejecutado la consulta Q5 de la cual se obtiene básicamente la información básica de los paquetes generados en el flujo.

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd
">
 <head>
  <variable name="flowIdVal"/>
  <variable name="seqNoVal"/>
  <variable name="tsVal"/>
  <variable name="pktLenVal"/>
  <variable name="dstHostVal"/>
  <variable name="dstPortVal"/>
 </head>
 <results distinct="false" ordered="true">
  <result>
   <binding name="flowIdVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">1</literal></binding
>
   <binding name="seqNoVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">1</literal></binding
>
   <binding name="tsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-09-
11T19:30:45.000001+02:00</literal></binding>
   <binding name="pktLenVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">512</literal></bindi
ng>
   <binding name="dstHostVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">localhost</literal></
binding>
   <binding name="dstPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">3000</literal></bind
ing>
  </result>
  <result>
   <binding name="flowIdVal"><literal
```

```
datatype="http://www.w3.org/2001/XMLSchema#integer">1</literal></binding
>
   <binding name="seqNoVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">2</literal></binding
>
   <binding name="tsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-09-
11T19:30:46+02:00</literal></binding>
   <binding name="pktLenVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">512</literal></bindi
ng>
   <binding name="dstHostVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">localhost</literal></
binding>
   <binding name="dstPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">3000</literal></bind
ing>
  </result>
  <result>
   <binding name="flowIdVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">1</literal></binding
>
   <binding name="seqNoVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">3</literal></binding
>
   <binding name="tsVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-09-
11T19:30:47.000001+02:00</literal></binding>
   <binding name="pktLenVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">512</literal></bindi
ng>
   <binding name="dstHostVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#string">localhost</literal></
binding>
   <binding name="dstPortVal"><literal
datatype="http://www.w3.org/2001/XMLSchema#integer">3000</literal></bind
ing>
  </result>
 </results>
</sparql>
```

Figura 4-5: Resultado de la traza de la aplicación de generación de paquetes.

4.2.2 Validación de compatibilidad

Como se ha comentado en todo el documento existe la necesidad de poder unificar diferentes bancos de datos sobre el mismo esquema de forma que se puedan realizar consultas conjuntas y obtener resultados. Para ello utilizando la arquitectura SPQR presentada en la sección 2.6 se han añadido las bases de datos semánticas y se han realizado las consultas, en este caso de Iperf para comprobar que se puede a partir de una misma consulta obtener datos de los 3 bancos de datos.

Para las consultas se han probado con la consulta Q1 y Q3, adjuntadas en el anexo A.

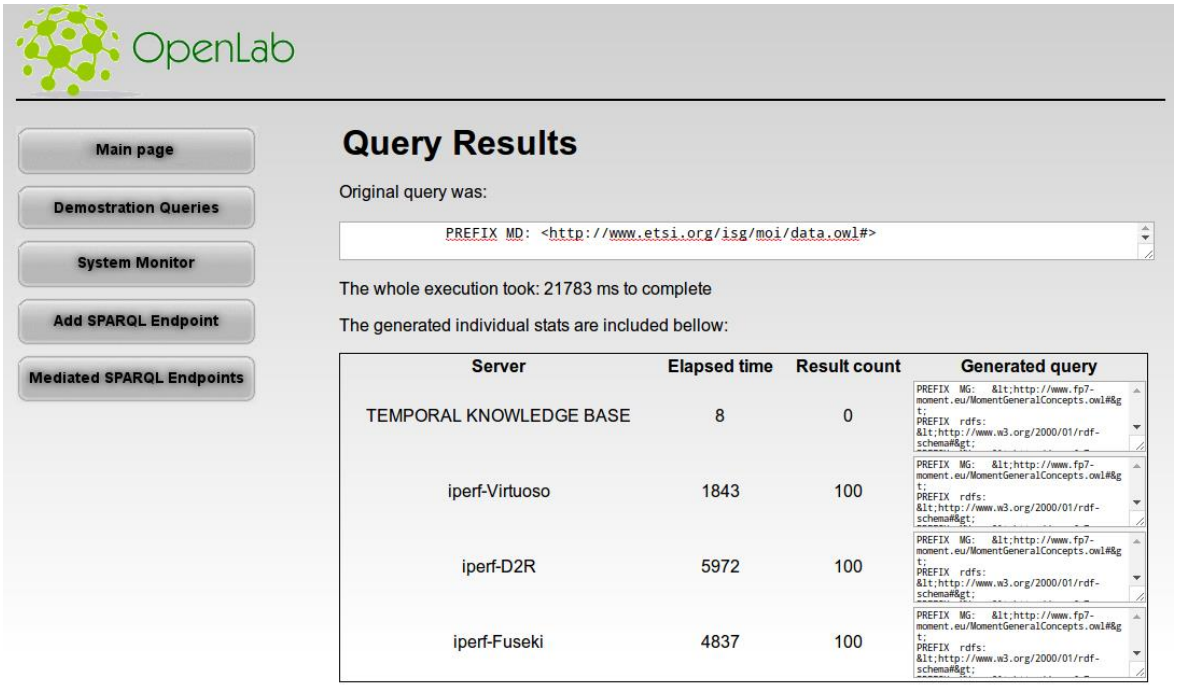


Figura 4-6: Captura de los resultados obtenidos en SPQR para la consulta Q1.

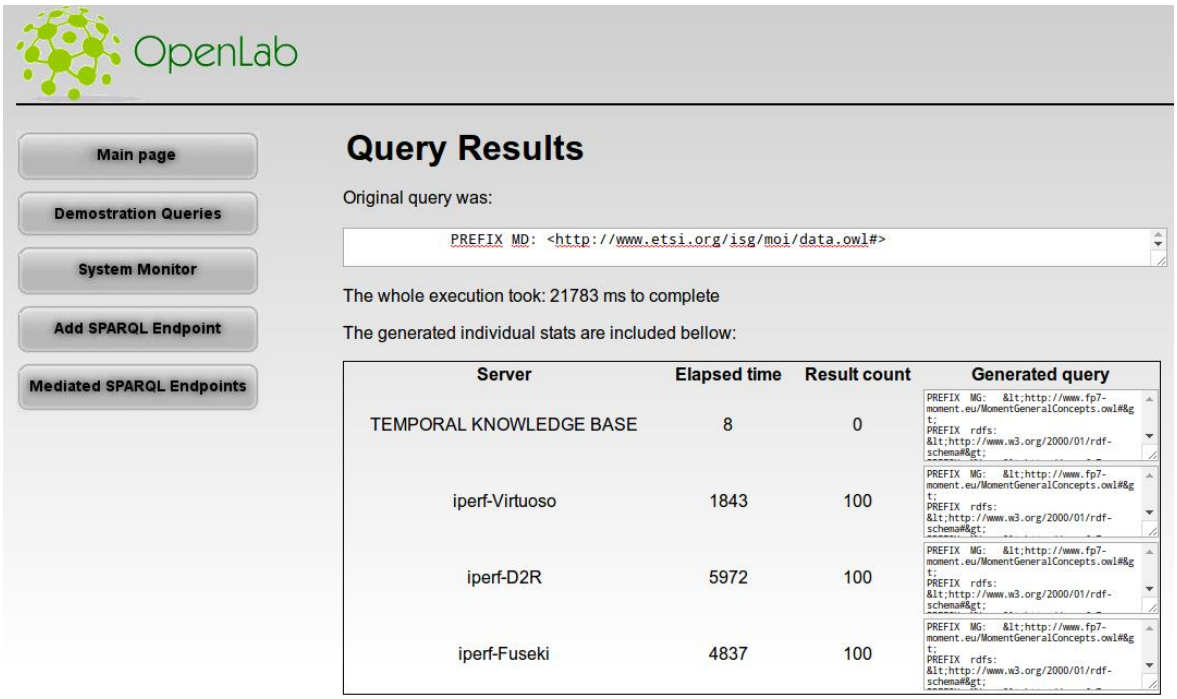


Figura 4-7: Captura de los resultados obtenidos en SPQR para la consulta Q3.

Por lo que se puede ver a la luz de las capturas todos ellos devuelven resultados dando en el caso de Virtuoso OpenLink una respuesta 3 veces más rápido que la solución D2R. El rendimiento de las distintas soluciones se estudiará más adelante en apartado 4.3.

4.3 Uso de backends y reporte de cliente/servidor OML

Como ya se comentó anteriormente el uso de las bases de datos relacionales obligan al usuario que obtiene un conjunto de medidas el esquema previo de cada banco de datos y tener que consultar por separado por lo que se desarrolló la idea de la ontología ETSI MOI como método de estandarización de los esquemas de los bancos de datos.

Aunque se utilizó la aproximación de D2R que permitía realizar una correspondencia entre conceptos de la ontología y campos de la base de datos. Para poder consultar datos desde el ámbito de la ontología este produce una latencia debido a que por cada consulta debe mantener el esquema de las tablas y realizar una transformación a SQL. Siendo en algunos casos no óptima para las bases de datos relacionales.

Esto propició estudiar la posibilidad de trabajar directamente con bases de datos semánticas de forma que se pueda eliminar el retardo producido por el servidor D2R. Por ello se ha propuesto la extensión de una de las bibliotecas más utilizadas para el reporte de medidas como es OML con la necesidad implícita de utilizar de sentencias de inserción semánticas, lo que obligó a utilizar el estándar SPARQL/UPDATE ya mencionado anteriormente.

En este apartado se analizará hasta qué punto la extensión de OML puede llegar a resultar rentable en términos de rendimiento y las ventajas que ello aporta.

4.3.1 Ventajas del uso de reporte realizado

Para poder analizar las ventajas en rendimiento del uso de la biblioteca extendida se debe comprobar qué ocurre a la hora de realizar cada inserción en cada tipo de base de datos. Para ello se ha medido el tiempo de cada inserción siendo la misma máquina la que tiene el servidor OML y el motor de base de datos de forma que el tiempo de una inserción se define como el tiempo que tarda en generarse la sentencia en el servidor OML, enviarla al motor de base de datos, ejecutarla y la espera de la confirmación afirmativa de que ha sido insertada en la base de datos.

Teniendo en cuenta esto se ha realizado un experimento con la aplicación Iperf donde se ha calculado el tiempo acumulado en milisegundos por cada 1000 inserciones de la medida del ancho de banda.

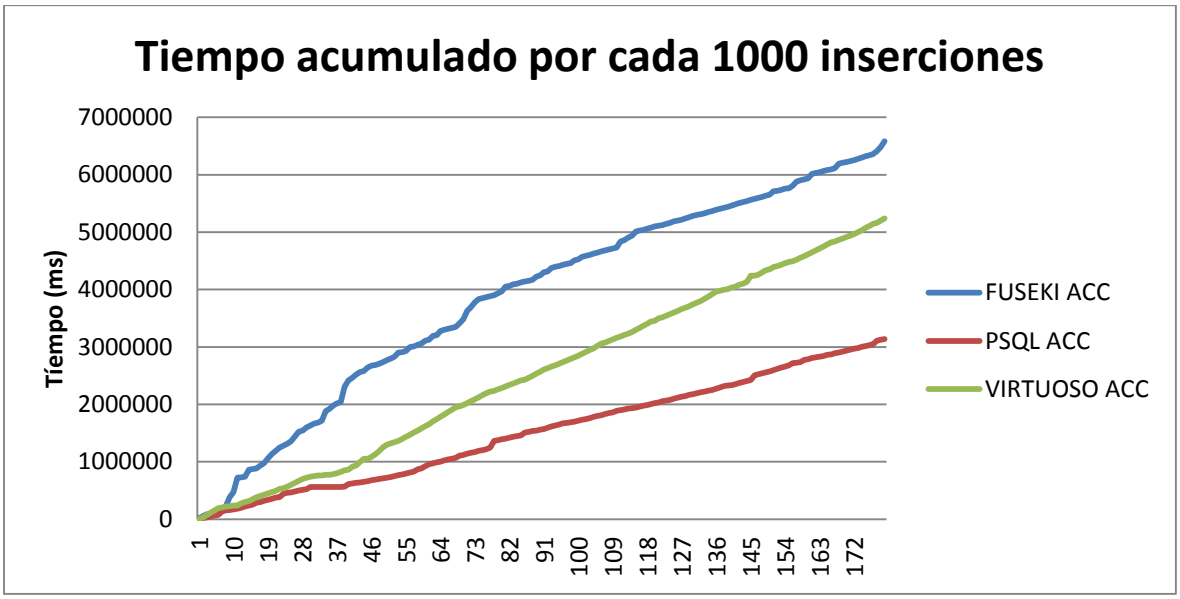


Figura 4-8: Grafica que representa el tiempo acumulado por cada 1000 inserciones.

Como se puede comprobar en la gráfica se tiene que claramente las inserciones en la base de datos de PostgreSQL requieren en total una menor cantidad de tiempo, en concreto, la mitad que en Fuseki y 3/5 partes que en Virtuoso. Lo que hacen la nueva solución en principio peor que D2R+PostgreSQL es natural debido a que las inserciones semánticas contienen más información que las relacionales.

Pero a diferencia de las inserciones, donde la biblioteca Semantic-OML da peor rendimiento. Es en la parte de consultas cuando se obtienen mejores tiempos. Debido a como se indicó que la solución de D2R+PostgreSQL tiene que mantener un esquema conceptual y no forma parte de la base de datos. Lo que lleva a que las consultas obtengan peores tiempos que en una base de datos semántica.

La primera prueba se realiza al consultar las direcciones IP implicadas en diferentes medidas Iperf, es decir, la consulta Q1 del anexo.

Q1	100 tuplas (arranque en frío)	100 tuplas
PostgresSQL+D2R	5972ms	1827ms
Fuseki+TDB	4837ms	1374ms
Virtuoso+TDB	1843ms	102ms

La segunda prueba que se realiza es la consulta Q3 del anexo obteniendo los siguientes resultados.

Q3	1K tuplas (arranque en frío)	1K tuplas	60K tuplas (arranque en frío)	60K tuplas	1M tuplas
PostgresSQL+D2R	2:43.36 (mm:s)	2:21.2 (mm:s)	28:45.8 (mm:s)	28:20.6 (mm:s)	- (hh:mm:s)
Fuseki+TDB	3:12.1 (mm:s)	2:58.9 (mm:ss)	29:34.9 (mm:s)	29:19.8 (mm:s)	11:20:00.0 (hh:mm:s)
Virtuoso+TDB	1:43.6 (mm:s)	0:58.9 (mm:ss)	23:32.4 (mm:s)	17:52.3 (mm:s)	9:30:00.0 (hh:mm:s)

Cuando se ejecuta la consulta Q3 para 1M de tuplas en el caso de PostgreSQL+D2R tanto cuando se pone a la aplicación java 1, 2 y 3 GB de memoria, aparte de tardar en torno a horas, da un fallo de falta de memoria. En el caso de Virtuoso se obtiene el resultado en 9 horas y 30 minutos aproximadamente y en caso de Fuseki se obtiene en 11 horas y 20 minutos aproximadamente. Cabe destacar que Virtuoso se configuró para 2GB en la última prueba y Fuseki con 3GB ya que con 2GB fallaba de igual forma que PostgreSQL+D2R.

Para Trace se ejecutó la consulta Q4 (sin límite de resultados obtenidos) adjuntada en el anexo A, donde se obtuvieron los siguientes resultados.

Q4 (Trace)	1K tuplas (arranque en frío)	1K tuplas	5K tuplas (arranque en frío)	5K tuplas
PostgresSQL+D2R	5:54.16 (mm:s)	5:34.7 (mm:s)	12:35.4 (mm:s)	12:24.6 (mm:s)
Fuseki+TDB	1:38.1 (mm:s)	1:12.9 (mm:ss)	3:24.5 (mm:s)	3:11.2 (mm:s)
Virtuoso+TDB	1:13.6 (mm:s)	0:46.0 (mm:ss)	1:56.7 (mm:s)	1:23.3 (mm:s)

Donde junto a la consulta Q3 se puede extraer que cuando se introduce que cuando se realiza una consulta simple en términos de restricciones pero que únicamente requiere recuperar los resultados la solución de D2R puede resultar beneficiosa. En parte debido porque el motor PostgreSQL tiene más tiempo de desarrollo por lo que está más optimizado. Además, según aumenta la cantidad de triplas y las restricciones de la consulta D2R no consigue formar una consulta SQL óptima por lo que se concluye en un detrimento del rendimiento.

Para otg2 se ejecutó la consulta Q5 (sin límite) adjuntadas en el anexo A, donde se obtuvo el siguiente resultado:

Q5 (otg2/otr2)	1K tuplas (arranque en frío)	1K tuplas	5K tuplas (arranque en frío)	5K tuplas
PostgresSQL+D2R	1:08.1 (mm:s)	1:01.7 (mm:s)	2:02.5 (mm:s)	1:55.8 (mm:s)
Fuseki+TDB	1:25.9 (mm:s)	1:12.4 (mm:ss)	2:14.4 (mm:s)	2:03.4 (mm:s)
Virtuoso+TDB	0:54.8 (mm:s)	0:47.3 (mm:ss)	1:32.4 (mm:s)	0:57.7 (mm:s)

En este caso se puede visualizar que al ser una consulta simple, es decir, para PostgreSQL solo debe operar con una tabla D2R consigue mejor rendimiento que Fuseki, tal y como se indicó en las conclusiones de la consulta Q4.

4.3.2 Ventajas de Fuseki

Del apartado anterior se desprende que para bases de datos pequeñas da buen resultado teniendo en cuenta que según incrementa la cantidad de triplas PostgreSQL+D2R da fallos de memoria insuficiente. Cuando el tamaño de la base de datos aumenta esta solución decrementa en rendimiento respecto a D2R lo que hace que para bases de datos demasiado grandes no sea rentable utilizar esta solución.

Esto puede ser debido a diversos factores, algunas publicaciones apuntan a que los motores semánticos aún no están tan optimizados como los relacionales [40] lo que hace que en ciertos casos PostgreSQL+D2R funcione mejor que Fuseki+TDB aunque se ha visto una mejora de rendimiento con el tiempo y sobretodo Fuseki ha conseguido mostrar el resultado que no consiguió D2R.

4.3.3 Ventajas de Virtuoso

Si se comprara Virtuoso OpenLink respecto a Fuseki y D2R se obtiene que es la solución con mejor rendimiento además de incluir algunas mejoras propias del motor de Virtuoso de las cuales para las pruebas de validación se desactivaron el mecanismo que tiene de caché de consultas o la partición de la base de datos en diversas bases de datos a las cuales preguntar por partes y luego unir resultados. Esto y demás mejoras son todas parametrizables desde el fichero de configuración siendo todo gestionado por el motor.

Además de lo anterior Virtuoso tiene un sistema de gestión de roles y permisos sobre inserciones de forma que un usuario puede escribir en un grafo semántico pero no en otro lo que añade ciertas ventajas desde el ámbito de la seguridad.

4.4 Ventajas respecto a D2R

De las pruebas realizadas se desprende que para el caso de bases de datos pequeñas el rendimiento obtenido es inapreciable, cuando el tamaño de la base de datos aumenta aunque el rendimiento de las consultas de las bases de datos semánticas decrementa se aprecia que en el caso de Virtuoso al permitir la partición de la consulta en subconjuntos de datos para posterior unión de los datos que junto con mecanismos de caché y prefijos en las tablas hace que pueda competir contra el motor D2R en rendimiento dando mejores resultados que este último.

Además hay que resaltar que cuando la consulta tiene un alto conjunto de restricciones D2R no consigue obtener la consulta más optimizada por lo que se obtiene un decremento notable del rendimiento.

Cabe destacar que, además de lo anterior, Virtuoso tiene un buffer de tamaño configurable para el almacenamiento temporal de las inserciones lo que se puede apreciar en la gráfica de los tiempos acumulados de las inserciones semánticas donde al principio sufre un aumento menor a cuando se ha ido llenando el buffer de memoria.

Además si se añade a esto la validación de conceptos de la ontología ETSI MOI aportado por la extensión semántica de OML scaffold, hace que tenga que ser tomado en cuenta la solución de utilizar la biblioteca Semantic-OML.

4.5 Conclusiones

A la luz de las pruebas realizadas, el desarrollo de Semantic-OML nos permite obtener una mejoría en el rendimiento de las consultas semánticas que siguen la estructura de la ontología ETSI MOI. Quizá es un problema en algunos casos el tema de que las inserciones sea algo lentas pero con el mecanismo de buffering de Virtuoso puede ser solucionado en la mayoría de estos.

Además, la ampliación de OML scaffold que genera el esqueleto permite una validación conceptual indicando en caso de error al menos una sugerencia de cambio al desarrollador tanto en el programa cliente Semantic-OML como a la hora de generar el código.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

En este Trabajo Fin de Master se ha contribuido doblemente. En primera instancia al proyecto europeo OpenLab, donde entre sus objetivos está la unificación de repositorios de medidas de red de forma que al permitir que una librería utilizada en muchos de estos bancos de pruebas al utilizar la extensión semántica permite la unificación de los repositorios de medidas bajo el manto de la ontología. En segunda instancia, gracias a esta ampliación han aparecido nuevos casos de uso para la ontología, lo que permitirá una mayor extensión y mejoría de esta teniendo en cuenta las nuevas aplicaciones que utilicen la biblioteca Semantic-OML.

Además, la facilidad de la realización de la correspondencia semántica permite que los técnicos que no están familiarizados con las ontologías puedan trabajar con ellas de forma que no tengan que conocerlas en detalle, sino únicamente que existen unos conceptos de medidas de red.

También indicar que este proyecto me ha permitido realizar varias contribuciones, no solo al proyecto OpenLab, la cual propició mi publicación en TridentCom [2] adjuntada en el anexo C, sino también la liberación del software desarrollado al proyecto OML liderado por NICTA.

Finalmente, el trabajo de poder ampliar una biblioteca que ya está en uso ha sido un trabajo ampliamente didáctico. Me ha permitido aprender mucho acerca de la adaptación de una biblioteca a unas necesidades particulares dentro de las propias de un proyecto europeo y la satisfacción de poder llegar a contribuir a éste.

5.2 Trabajo futuro

Existen múltiples líneas de trabajo, aunque las más directa es adaptar la mayor cantidad de programas a esta librería para recolectar la mayor cantidad posible de casos de uso y realizar aportaciones de peso de diversos casos de uso útiles y prácticos permitiendo así poder ampliar la ontología de ETSI MOI.

Otra línea posible, sería analizar si se puede generar un sistema de consultas preparadas en el lado del servidor que permita almacenar las consultas y generar insercciones preparadas nativas de forma que se disminuya la latencia por inserción. También, de cara a obtener un mejor rendimiento pero esta vez de parte de las consultas, se podría estudiar qué ocurre cuando se piden datos con pocas restricciones y muchas tuplas, ya que en este caso dan mejores resultados soluciones que utilicen motores SQL.

Finalmente, aunque menos directa, podría ser trabajar en el sistema de validación de la definición semántica para poder obtener mejores sugerencias o incluso realizar un sistema (posiblemente online) en que realizar las definiciones donde se permita reportar las necesidades y a partir de las aplicaciones de otras herramientas se pueda realizar sugerencias al programador.

Referencias

[1] http://www.ict-openlab.eu/

[2] Jorge E. López de Vergara, Víctor Acero, Mario Poyato, Javier Aracil, "A semantic interface for OpenLab network measurement infrastructures", Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TridentCom'2012. Thessaloniki, Greece; June 2011

[3] Jorge Enrique López de Vergara Méndez, Director: Víctor Abraham Villagrá González, "Especificación de Modelos de Información de Gestión de Red Integrada Mediante el Uso de Ontologías y Técnicas de Representación del Conocimiento". Tesis Doctoral, Universidad Politécnica de Madrid, 2003.

[4] "Measurement Ontology for IP traffic (MOI); Report on information models for IP traffic measurement". ETSI DSG/MOI-0010, May 2010.

[5] "Measurement Ontology for IP traffic (MOI); Requirements for IP traffic measurement ontologies development", ETSI GS MOI 002 V1.1.1, July 2012.

[6] Measurement Ontology for IP traffic (MOI); IP traffic measurement ontologies architecture," ETSI GS MOI 003 V1.1.2 January 2013.

[7] http://oml.mytestbed.net/projects/oml/wiki

[8] Manpreet Singh, Max Ott, Ivan Seskar, and Pandurang Kama, "ORBIT measurements framework and library (OML): Motivations, design, implementation, and features," in TridentCom 2005, Feb. 2005

[9] http://www.w3.org/TR/sparql11-update/

[10] http://jena.apache.org/documentation/serving_data/

[11] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/

[12] http://d2rq.org/d2r-server

[13] http://parliament.semwebcentral.org/

[14] http://jena.apache.org/documentation/tdb/

[15] Especificación de una Ontología de Medidas para Internet. Trabajo final de grado de Daniel Michaud.

[16] http://www.w3.org/2007/OWL/wiki/PrimerExampleTurtle

[17] http://protege.stanford.edu/

[18] http://sweet.jpl.nasa.gov/ontology/units.owl

[19] http://www.datcat.org/

[20] https://mytestbed.net/projects/oml/wiki/Quick_Start_Tutorial

[21] http://oml.mytestbed.net/doc/oml/latest/oml2-scaffold.1.html

[22] http://d2rq.org/generate-mapping

[23] https://mytestbed.net/projects/omlapp/wiki/Iperf

[24] http://sourceforge.net/projects/joseki/

[25] Chris Bizer, Andreas Schultz. The Berlin SPARQL Benchmark. Int. J. Semantic Web Inf. Syst. 5(2): 1-24 (2009)

[26] https://github.com/ruby-rdf/rdf

[27] https://www.ruby-toolbox.com/projects/text

[28] http://www.catalysoft.com/articles/StrikeAMatch.html

[29] http://www.w3.org/TR/sparql11-query/

[30] http://www.w3.org/TR/rdf-sparql-query/

[31] http://www.w3.org/TR/sparql11-overview/

[32] http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#dfn-URI-reference

[33] http://db-engines.com/
[34] http://db-engines.com/en/ranking_definition
[35] http://jena.apache.org/index.html
[36] https://iperf.fr/
[37] http://research.wand.net.nz/software/libtrace.php
[38] http://www.tcpdump.org/pcap.html
[39] https://glassfish.java.net/es/
[40] Christian Bizer, Andreas Schultz, "The Berlin SPARQL Benchmark" Freie Universität Berlin, Web-based Systems Group, Garystr. 21, 14195 Berlin, Germany
[41] J.E. López de Vergara, J. Aracil, "Measurements and Measurement Tools in OpenLab: Use Cases with Measurement Data Ontologies", Proc. EULER Workshop in Measurement and Measurement Tools, Aalborg, Denmark, May 2012. Lecture Notes in Computer Science, Vol. 7586, pp. 155-170, Springer, ISSN 0302-9743, doi:10.1007/978-3-642-41296-7_10
[42] Alfredo Salvador, Jorge E. López de Vergara, Alvaro Katsu, Javier Aracil, "SPQR: SPARQL pre-Processor and Query Rewriter for a semantic access to multiple heterogeneous network measurements", Proc. 1st IFIP/IEEE International Workshop on Knowledge Management for Future Services and Networks (KMFSAN 2010), Osaka, Japan, 23 april 2010.
[43] Gruber, Thomas R. "A translation approach to portable ontology specifications". Knowledge acquisition 5.2 (1993): 199-220.
[44] http://www.w3.org/RDF/
[45] http://www.w3.org/TR/rdf-schema/
[46] V.I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady (1966). Volume: 10, Issue: 8, Pages: 707-710.
[47] "How to Strike a Match". Original article of the algorithm by Simon White.
[48] http://www.w3.org/TR/sparql11-http-rdf-update/
[49] http://www.openrdf.org/

Glosario

SPARQL	Protocol and RDF Query Languaje
RDF	Resource Description Framework
SPQR	SPARQL pre-Processor and Query Rewriter
SMR	Semantic Metadata Registry
SQL	Structured Query Language
Semantic-OML	Semantic OMF Measurement Library
XML	eXtensible Markup Language
ETSI	European Telecommunications Standards Institute
MOI	Measurement Ontology for IP traffic
ETSI	European Telecommunications Standard Institute
OMF	cOntrol and Management Framework
OML	OMF Measurement Library
MP	Measurement Point
MOMENT	Monitoring and Measurement in the Next generation Technologies
CSV	Comma-Separated Values
JDBC	Java DataBase Connectivity
GPL	General Public License
KPI	Key Performance Indicator
IRI	Internationalized Resource Identifier

Anexos

A Consultas

Nombre	Consulta

Q1	SELECT ?SourceIPValue ?DstIPValue
	WHERE {
	?Iperf rdf:type MD:Iperf ;
	MD:hasMetricAttributes ?IdObj ;
	MD:hasMetricAttributes ?SourceObj ;
	MD:hasMetricAttributes ?DstObj .
	?IdObj rdf:type MD:id ;
	MD:MeasurementDataValue ?IdValue .
	?SourceObj rdf:type MD:SourceIP ;
	MD:SourceIPValue ?SourceIPValue .
	?DstObj rdf:type MD:DestinationIP ;
	MD:DestinationIPValue ?DstIPValue .
	}

Q2	SELECT ?SourceIPValue ?DstIPValue MIN(?capacityValue) AVG(?capacityValue) MAX(?capacityValue) WHERE { ?Iperf rdf:type MD:Iperf ; MD:hasMetricAttributes ?IdObj ; MD:hasMetricAttributes ?SourceObj ; MD:hasMetricAttributes ?DstObj . ?IdObj rdf:type MD:id ; MD:MeasurementDataValue ?IdValue . ?SourceObj rdf:type MD:SourceIP ;

	MD:SourceIPValue ?SourceIPValue . ?DstObj rdf:type MD:DestinationIP ; MD:DestinationIPValue ?DstIPValue . ?Iperf rdf:type MD:Iperf ; MD:hasMetricAttributes ?IdObj ; MD:hasMetricAttributes ?capObj . ?IdObj2 rdf:type MD:id ; MD:MeasurementDataValue ?IdValue2 . ?capObj rdf:type MD:CapacityMeasurement ; MD:CapacityMeasurementValue ?capacityValue . } GROUP BY ?SourceIPValue ?DstIPValue
Q3	SELECT ?SourceIPValue ?DstIPValue MIN(?startTimeValue) MAX(?endTimeValue) MIN(?capacityValue) AVG(?capacityValue) MAX(?capacityValue) WHERE { ?Iperf rdf:type MD:Iperf ; MD:hasMetricAttributes ?IdObj ; MD:hasMetricAttributes ?startTime ; MD:hasMetricAttributes ?endTime ; MD:hasMetricAttributes ?SourceObj ; MD:hasMetricAttributes ?DstObj . ?IdObj rdf:type MD:id ; MD:MeasurementDataValue ?IdValue . ?SourceObj rdf:type MD:SourceIP ; MD:SourceIPValue ?SourceIPValue . ?DstObj rdf:type MD:DestinationIP ; MD:DestinationIPValue ?DstIPValue . ?Iperf rdf:type MD:Iperf ;

	MD:hasMetricAttributes ?ldObj ; MD:hasMetricAttributes ?capObj . ?ldObj2 rdf:type MD:id ; MD:MeasurementDataValue ?IdValue2 . ?capObj rdf:type MD:CapacityMeasurement ; MD:CapacityMeasurementValue ?capacityValue . ?startTime rdf:type MGC:TimeStamp ; MGC:startTime ?startTimeValue . ?endTime rdf:type MGC:TimeStamp ; MGC:endTime ?endTimeValue . } GROUP BY ?SourceIPValue ?DstIPValue
Q4	SELECT ?type ?IPidVal ?IPttlVal ?IPtsVal ?UPPtsVal ?IPproVal ?IPsrcVal ?UPPsrcPortVal ?IPdstVal ?UPPdstPortVal ?IPSizeVal ?UPPsizeVal # ---------- IP ---------- ?TraceIP a MD:TrainPacketMeasurement ; MD:hasMeasurementData ?id_ip ; MD:hasMeasurementData ?IPid ; MD:hasMeasurementData ?IPttl ; MD:hasMeasurementData ?IPpro ; MD:hasMeasurementData ?IPsrc ; MD:hasMeasurementData ?IPdst ; MD:hasMeasurementData ?IPSize ; MD:hasMeasurementData ?IPts . ?id_ip a MD:PacketIdentifier ; MD:PacketIdentifierValue ?idVal . ?IPid a MD:IpIdentifier ; MD:PacketAndTrainPropertiesValue ?IPidVal .

	?IPttl a MD:FinalTtlMeasurement ; MD:SimpleMeasurementValue ?IPttlVal . ?IPpro a MGC:TransportProtocol ; MGC:protocolNumber ?IPproVal . ?IPsrc a MD:SourceIP ; MD:SourceIPValue ?IPsrcVal . ?IPdst a MD:DestinationIP ; MD:DestinationIPValue ?IPdstVal . ?IPSize a MD:PacketSize ; MD:PacketSizeValue ?IPSizeVal . ?IPts a MGC:TimeStamp ; MGC:timestamp ?IPtsVal . # ---------- UDP or TCP ---------- ?TraceUPP a MD:TrainPacketMeasurement ; rdfs:label ?type ; MD:hasMeasurementData ?id_UPP ; MD:hasMetricAttributes ?UPPsrcPort; MD:hasMetricAttributes ?UPPdstPort; MD:hasMeasurementData ?UPPsize ; MD:hasMeasurementData ?tsUPP . ?id_UPP a MD:PacketIdentifier ; MD:PacketIdentifierValue ?idVal . ?UPPsrcPort a MD:SourcePort ; MD:SourcePortValue ?UPPsrcPortVal . ?UPPdstPort a MD:DestinationPort ; MD:DestinationPortValue ?UPPdstPortVal . ?UPPsize a MD:PacketSize ; MD:PacketSizeValue ?UPPsizeVal .

	?tsUPP a MGC:TimeStamp ; MGC:timestamp ?UPPtsVal . FILTER (?IPproVal = 6) # Get TCP packets (protocol 6) FILTER (?UPPdstPortVal == 443) # get HTTPS packets (protocol 443) } LIMIT 2
Q5	select DISTINCT ?flowIdVal ?seqNoVal ?tsVal ?pktLenVal ?dstHostVal ?dstPortVal WHERE { ?otg a MD:TrainPacketMeasurement ; MD:hasMeasurementData ?ts ; MD:hasMeasurementData ?flowId ; MD:hasMeasurementData ?seqNo ; MD:hasMeasurementData ?pktLen ; MD:hasMeasurementData ?dstHost ; MD:hasMeasurementData ?dstPort . ?ts a MD:Time ; MD:TimeValue ?tsVal . ?flowId a MD:TrainIdentifier ; MD:TrainIdentifierValue ?flowIdVal . ?seqNo a MD:PacketIdentifier ; MD:PacketIdentifierValue ?seqNoVal . ?pktLen a MD:PacketSize ; MD:PacketSizeValue ?pktLenVal . ?dstHost a MD:DestinationIP ; MD:DestinationIPValue ?dstHostVal . ?dstPort a MD:DestinationPort ; MD:DestinationPortValue ?dstPortVal .

	} LIMIT 3

B Documentación para la contribución con el proyecto OML

Motivation

There are many systems to monitor network traffic, providing measurements about delay, jitter, capacity, packet loss, etc. Most of them use different data structures, they provide it in different units and sometimes they use different algorithms. A common information model of network measurement parameters and units has to be agreed. Some reasons are, for instance, to clarify SLAs, to exchange network monitoring information, to mix information from several sources, or to develop complex monitoring systems.

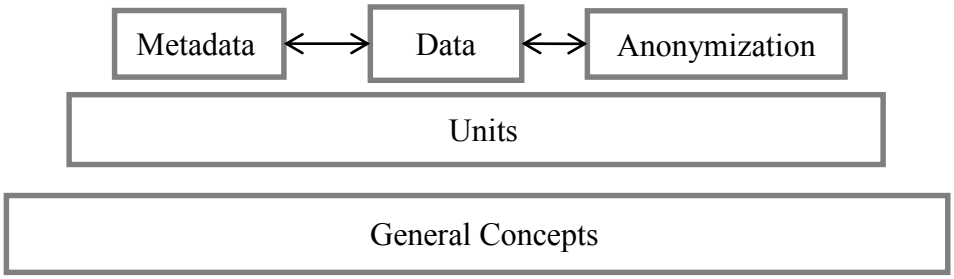
In this context, MOI (Measurement Ontology for IP Traffic) was created as an ETSI ISG (Industrial Specification Group) to enable the specification of an ontology for IP network traffic measurements. This ISG includes in its members network operators, SMEs, research centers and universities. The work done at MOI is much related to past EU FP7 projects: MOMENT, PRISM, NOVI and OpenLab [6].

Currently, it has completed three documents (Work Items):

- WI#1: Report on information models for IP traffic measurement [1].
- WI#2: Requirements for IP traffic measurement ontologies development [2]
- WI#3: Measurement Ontology for IP traffic (MOI) [3][5].

Finally, the ETSI MOI ontology architecture has the following parts:

- General concepts: a set of concepts in the network measurement domain.
- Units: The set of interpretable units used in network measurements.
- Metadata: It provides information about what, when was measured, who measured it, and where such measurement can be located.
- Data where each entry in a data source is mapped into a Measurement instance.
- Anonymization: identify the common language.



In the Openlab project context there were a large set of federated testbeds with similar concepts to other testbeds and network measurement platforms inside and outside of the projects. Some cases are ETOMIC, nmVO, TopHat, and Zabbix.

Furthermore, there were testbeds which use OML as their measurement report framework. Thus, OML is in our scope in the measurement data integration.

Measurement data integration

Previously to Semantic-OML the measurement data integration was performed with a D2RQ mapping between SQL schema and the ETSI MOI Ontology. To perform this, we developed the following architecture:

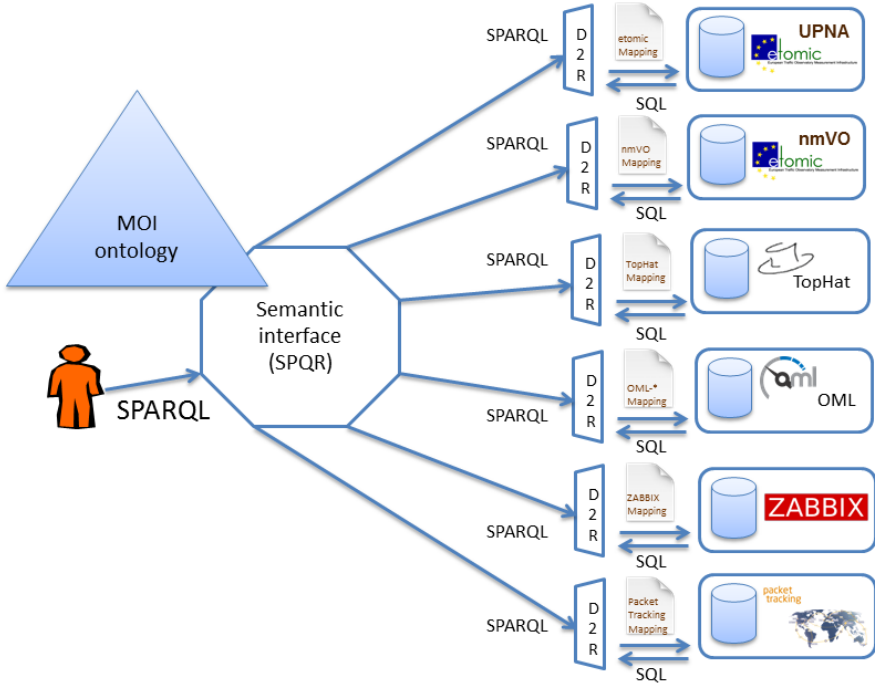


Image 1: Semantic architecture before Semantic-OML

In this use case the user sends a SPARQL query to a Web Service in our case (SPQR) connected to all semantic databases and his/her query is distributed to all D2R servers which perform the transformation between SPARQL and SQL, afterwards each semantic endpoint answers the queried data.

The main step to build this architecture after understanding SQL schema is to relate most tables and columns with MOI concepts, where each table (or subject) usually maps to a Measurement concept and each column maps to a MeasurementData. Facets of data such as units or data types are also defined in a D2R mapping file as the following image explains:

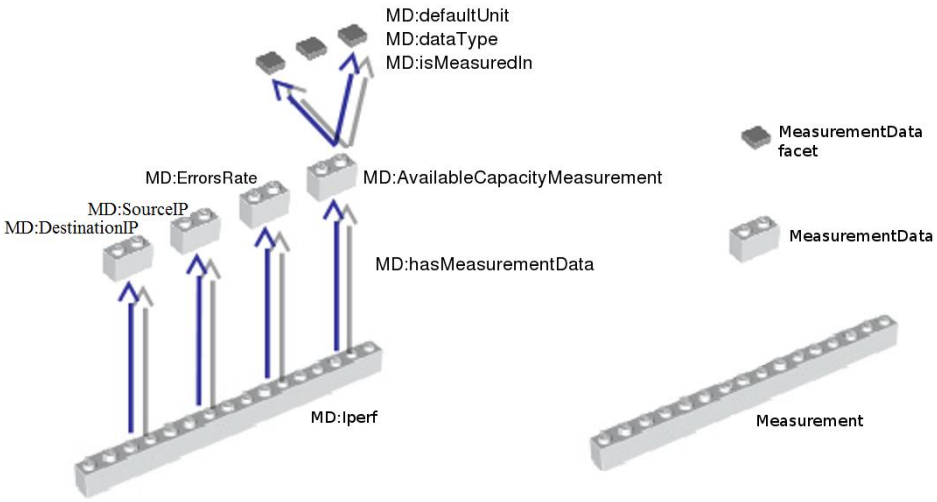


Image 2: Iperf mapping example.

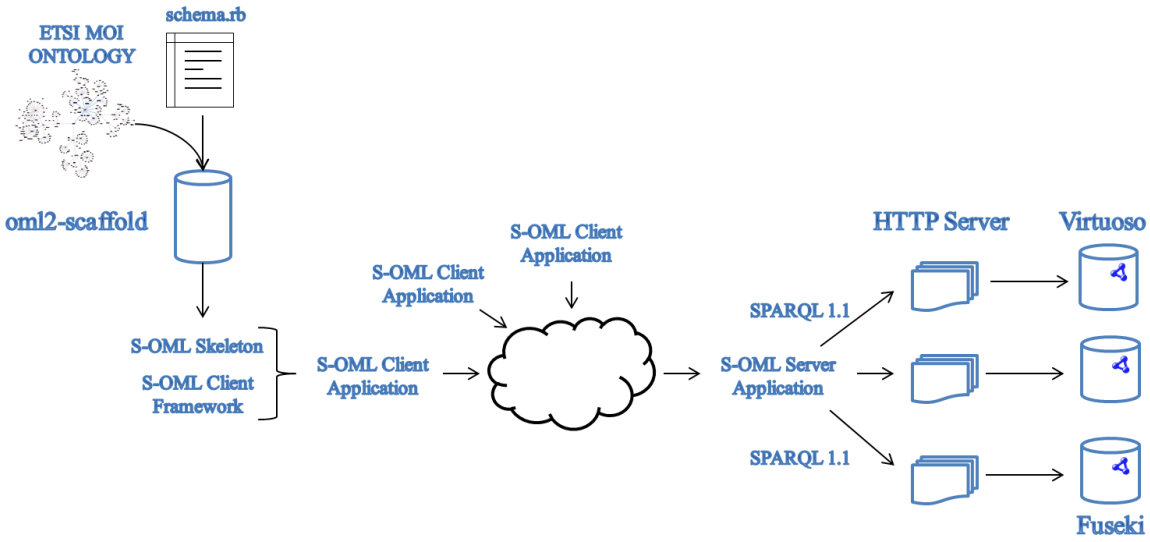
The main problem in this solution is that D2R uses a lot of memory and SQL queries are not optimized in all queries, such as joins when they are not necessary. Thus, we have defined this OML extension to get more performance with respect to the previous architecture and retrieve information faster than D2R solution where it is possible.

Semantic OML Extension (Semantic-OML)

The main idea is to add the concepts in the Measurement Point (MP) specification in order to communicate from OML Client application to OML Server the semantic schema and building the insert statement.

The semantic extension has three main parts: OML scaffold instrumentation extension, OML client extension and OML server add-on.

The following diagram shows the new semantic-OML architecture:



OML2-Scaffold instrumentation extension

As specified in OML documentation, oml2-scaffold generates the OML client skeleton from a schema definition wrote in Ruby. Thus, this is the best place where a Semantic-OML client application developer can define the semantic schema, relations and concept facets related with the ETSI MOI Ontology.

Therefore, this schema specification may be extended from OML definition as shown below:

```
app.defMeasurement(<title>){ |m|
    m.defMetric(<name>, <type>, <description>,
        [
            ['<Subject>','<Predicate>', '<Verb>'],
            ['<Subject>','<Predicate>', '<Verb>'],
            (…)
            ['<Subject>','<Predicate>', '<Verb>']
        ]
    )
    (…)
    m.defMetric(<name>, <type>, <description>,
        [
            (…)
        ]
    )
}
```

Where all hierarchy schema for a metric is defined from Measurement concept to MeasurementData concept and metric facets such as its default unit.

An example abstract definition for a source IP metric is:

```
m.defMetric('source_ip', :string, 'Source node ipv4',
    [
        ['MD:Iperf','MD:hasMetricAttibutes', 'MD:SourceIP'],
        ['MD:SourceIP','MD:SourceIPValue', '%value%'],
        ['MD:SourceIP','MD: defaultUnit', 'MU: ipv4dotted']
    ]
)
```

Where all triples are defined for the metric from Measurement to MeasurementData and its value, which matched by "*%value%"* string, and its appropriate unit.

Futhermore, this schema can be validated while oml-scaffold generates the Semantic-OML client skeleton with ETSI MOI Ontology concepts. This validation requires a Ruby version upper to 1.9.2 and "rdf/rdfxml" [4] plugin allowing a developer to check that all Ruby specification concepts are defined in the ETSI MOI Ontology.

Finally, if the developer has installed "text" [5] library, when a concept is incorrect the OML application shows a warning message with the best suggestion calculated in order to correct his/her specification. First suggestion is the concept with the minimum Levensthein distance, and it prevents commonly writing errors, and second suggestion is the white similar concept and prevents word ordering errors, for example, the undefined concept MD:IPSource matches with MD:SourceIP.

```
INFO oml2-scaffold V2.12.0pre.77-4992-dirty Copyright 2009-2014,
NICTA
INFO with semantic concepts validating (Copyright 2013-2014, HPCN -
UAM)
WARN Concept MD:IPSource is not in the Moment Ontology. Sugestions:
"MD:SourceIP" or "MD:SourceIP"
```

OML Client extension

This component only adds the generated oml2-scaffold skeleton into the header schema of the measurement point when a new schema is added into the experimentation.

This module has the following schema:

```
INFO    <domain>:<experiment>:<app>: New MS schema <1…n>
<measurement_name>
<name_metric>:<type_metric>:{<subject>|<verb>|<predicate>}{<subject>|<v
erb>|<predicate>} <name_metric>:<type_metric>…
```

This functionality only transfers the schema to the OML server. The following buffer is an example:

```
INFO    iperf_20:experiment_2:iperf: New MS schema 4 iperf_connection
pid:int32:{MD:Iperf|MD:hasMetricAttributes|MD:id}{MD:id|MD:MeasurementD
ataValue|%value%} connection_id:int32
local_address:string:{MD:Iperf|MD:hasMetricAttributes|MD:SourceIP}{MD:S
ourceIP|MD:SourceIPValue|%value%}{MD:SourceIP|MD:defaultUnit|MU:ipv4dot
ted}
local_port:int32:{MD:Iperf|MD:hasMetricAttributes|MD:SourcePort}{MD:Sou
rcePort|MD:SourcePortValue|%value%}
remote_address:string:{MD:Iperf|MD:hasMetricAttributes|MD:DestinationIP
}{MD:DestinationIP|MD:DestinationIPValue|%value%}{MD:DestinationIP|MD:d
efaultUnit|MU:ipv4dotted}
remote_port:int32:{MD:Iperf|MD:hasMetricAttributes|MD:DestinationPort}{
MD:DestinationPort|MD:DestinationPortValue|%value%}
```

OML Server Add-on

This component centralizes most add-ons in Semantic-OML where the endpoint database is an RDFendpoint that supports SPARQL/Update 1.1. defined by W3C [7]. In our tests we have used HTTP interface because it is the most standard solution, and Virtuoso Openlink and Fuseki server with TDB triple stores as endpoints, which are currently used in most semantic web developments.

FUSEKI

Fuseki is the best solution if the Semantic-OML developer wants to choose the fastest deployment in order to use a small store which is easily configured and does not use many computation resources.

This solution builds insertions structure and it has the following details:

- Measurement identification is generated with *struuid* function defined in SPARQL/Update 1.1 by endpoint because HTTP interface does not have transactional insertions.
- Inserts all triples into a graph with a URI defined by server IP and port and domain.
- Default namespace defined is "/ds".
- This solution does not have security options by default in the endpoint. For this, configure Jetty server or run TDB database over Apache server.

An example to execute Fuseki solution is:

```
oml2-server –b fuseki –sem-namespace /ds –sem-host 127.0.0.1 –sem-port 3030
```

VIRTUOSO OPENLINK

Virtuoso Openlink is the best solution if the Semantic-OML developer wants to choose the most performed solution and use the semantic solution in order to use a big store.

This solution builds insertions structure and it has the following details:

- Measurement identification is generated with a function defined in Virtuoso documentation unique generated by endpoint because HTTP interface does not have transactional insertions.
- Inserts all triples into a graph with a URI defined by server IP and port and domain.
- This solution has roles and security implemented and is only necessary to configure Virtuoso server roles as defined in the documentation.

An example to execute Virtuoso solution is:

```
oml2-server –b virtuoso –sem-user dba –sem-pass dba –sem-host 127.0.0.1 –sem-port 8890
```

Iperf example

First of all, it is necessary to define the semantic schema and generate application code:

```
app.defMeasurement("application"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('version', :string, 'Iperf version',
        [['MD:Iperf','MD:hasMeasurementData','MD:ToolVersion'],
        ['MD:ToolVersion','MD:ToolVersionValue','%value%']
    ])
    m.defMetric('cmdline', :string, 'Iperf invocation command line',
        [['MD:Iperf','MD:hasMeasurementData','MD:Arguments'],
        ['MD:Arguments','MD:ArgumentsValue','%value%']
    ])
```

```
    m.defMetric('starttime_s', :integer, 'Time the application was received (s)',
        [['MD:Iperf','MD:hasMeasurementData','MD:Arguments'],
        ['MD:Arguments','MD:ArgumentsValue','%value%']
    ])
    m.defMetric('starttime_us', :integer, 'Time the application was received (us)',
        [['MD:Iperf','MD:hasMeasurementData','MD:Arguments'],
        ['MD:Arguments','MD:ArgumentsValue','%value%']
    ])
  }

  app.defMeasurement("settings"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('server_mode', :integer, '1 if in server mode, 0 otherwise')
    m.defMetric('bind_address', :string, 'Address to bind',
        [['MD:Iperf','MD:hasMetricAttributes','MD:NodeIP'],
        ['MD:NodeIP','MD:NodeIPValue','%value%'],
        ['MD:NodeIP','MD:defaultUnit','MU:ipv4dotted']
    ])
    m.defMetric('multicast', :integer, '1 if listening to a Multicast group',
        [['MD:Iperf','MD:hasMeasurementData','MD:GenericParameter'],
        ['MD:GenericParameter','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('multicast_ttl', :integer, 'Multicast TTL if relevant',
        [['MD:Iperf','MD:hasMetricAttributes','MD:FirstTtlMeasurement'],
        ['MD:FirstTtlMeasurement','MD:FirstTtlMeasurementValue','%value%']
    ])
    m.defMetric('transport_protocol', :integer, 'Transport protocol (IANA number)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TransportProtocol'],
        ['MGC:TransportProtocol','MGC:protocolNumber','%value%']
    ])
    m.defMetric('window_size', :integer, 'TCP window size',
        [['MD:Iperf','MD:hasMeasurementData','MD:Arguments'],
        ['MD:Arguments','MD:ArgumentsValue','%value%']
    ])
    m.defMetric('buffer_size', :integer, 'UDP buffer size',
        [['MD:Iperf','MD:hasMeasurementData','MD:Arguments'],
        ['MD:Arguments','MD:ArgumentsValue','%value%']
    ])
  }

  app.defMeasurement("connection"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('connection_id', :integer, 'Connection identifier (socket)')
    m.defMetric('local_address', :string, 'Local network address',
        [['MD:Iperf','MD:hasMetricAttributes','MD:SourceIP'],
        ['MD:SourceIP','MD:SourceIPValue','%value%'],
        ['MD:SourceIP','MD:defaultUnit','MU:ipv4dotted']
    ])
    m.defMetric('local_port', :integer, 'Local port',
        [['MD:Iperf','MD:hasMetricAttributes','MD:SourcePort'],
        ['MD:SourcePort','MD:SourcePortValue', '%value%']
    ])
    m.defMetric('remote_address', :string, 'Remote network address',
        [['MD:Iperf','MD:hasMetricAttributes','MD:DestinationIP'],
        ['MD:DestinationIP','MD:DestinationIPValue','%value%'],
        ['MD:DestinationIP','MD:defaultUnit','MU:ipv4dotted']
    ])
    m.defMetric('remote_port', :integer, 'Remote port',
        [['MD:Iperf','MD:hasMetricAttributes','MD:DestinationPort'],
        ['MD:DestinationPort','MD:DestinationPortValue', '%value%']
    ])
  }

  app.defMeasurement("transfer"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
```

```
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('connection_id', :integer, 'Connection identifier (socket)')
    m.defMetric('begin_interval', :double, 'Start of the averaging interval (Iperf
timestamp)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TimeStamp'],
        ['MGC:TimeStamp','MGC:startTime','%value%'],
        ['MGC:TimeStamp','MD:defaultUnit','MU:second']
    ])
    m.defMetric('end_interval', :double, 'End of the averaging interval (Iperf
timestamp)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TimeStamp'],
        ['MGC:TimeStamp','MGC:endTime','%value%'],
        ['MGC:TimeStamp','MD:defaultUnit','MU:second']
    ])
    m.defMetric('size', :uint64, 'Amount of transmitted data
        [Bytes]')
    m.defMetric('capacity', :double, 'Capacity of the link
        [Bytes per second]',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:CapacityMeasurement'],
        ['MD:CapacityMeasurement','MD:CapacityMeasurementValue','%value%'],
        ['MD:CapacityMeasurement','MD:defaultUnit','MU:Bytepersec']
    ])
  }

  app.defMeasurement("losses"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('connection_id', :integer, 'Connection identifier (socket)')
    m.defMetric('begin_interval', :double, 'Start of the averaging interval (Iperf
timestamp)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TimeStamp'],
        ['MGC:TimeStamp','MGC:startTime','%value%'],
        ['MGC:TimeStamp','MD:defaultUnit','MU:second']
    ])
    m.defMetric('end_interval', :double, 'End of the averaging interval (Iperf
timestamp)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TimeStamp'],
        ['MGC:TimeStamp','MGC:endTime','%value%'],
        ['MGC:TimeStamp','MD:defaultUnit','MU:second']
    ])
    m.defMetric('total_datagrams', :integer, 'Total number of datagrams',
        [['MD:Iperf','MD:hasMeasurementData','MD:PacketsCount'],
        ['MD:PacketsCount','MD:PacketsCountValue','%value%']
    ])
    m.defMetric('lost_datagrams', :integer, 'Number of lost datagrams',
        [['MD:Iperf','MD:hasMeasurementData','MD:ErrorsRate'],
        ['MD:ErrorsRate','MD:ErrorsRateValue','%value%']
    ])
  }

  app.defMeasurement("jitter"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('connection_id', :integer, 'Connection identifier (socket)')
    m.defMetric('begin_interval', :double, 'Start of the averaging interval (Iperf
timestamp)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TimeStamp'],
        ['MGC:TimeStamp','MGC:startTime','%value%'],
        ['MGC:TimeStamp','MD:defaultUnit','MU:second']
    ])
    m.defMetric('end_interval', :double, 'End of the averaging interval (Iperf
timestamp)',
        [['MD:Iperf','MD:hasMetricAttributes','MGC:TimeStamp'],
        ['MGC:TimeStamp','MGC:endTime','%value%'],
        ['MGC:TimeStamp','MD:defaultUnit','MU:second']
    ])
    m.defMetric('jitter', :double, 'Average jitter
```

```
        [ms]',
        [['MD:Iperf','MD:hasMetricAttributes','MD:DelayVariationMeasurement'],
        ['MD:DelayVariationMeasurement','MD:DelayVariationMeasurementValue','%value%'],
        ['MD:DelayVariationMeasurement','MD:defaultUnit','MU:millisecond']
    ])
  }

  app.defMeasurement("packets"){ |m|
    m.defMetric('pid', :integer, 'Main process identifier',
        [['MD:Iperf','MD:hasMetricAttributes', 'MD:id'],
        ['MD:id','MD:MeasurementDataValue','%value%']
    ])
    m.defMetric('connection_id', :integer, 'Connection identifier (socket)')
    m.defMetric('packet_id', :integer, 'Packet sequence number for datagram-oriented
protocols',
        [['MD:Iperf','MD:hasMeasurementData','MD:PacketIdentifier'],
        ['MD:PacketIdentifier','MD:PacketIdentifierValue','%value%']
    ])
    m.defMetric('packet_size', :integer, 'Packet size',
        [['MD:Iperf','MD:hasMeasurementData', 'MD:PacketSize'],
        ['MD:PacketSize','MD:PacketSizeValue','%value%'],
        ['MD:PacketSize','MD:defaultUnit','MU:Byte']
    ])
    m.defMetric('packet_time_s', :integer, 'Time the packet was processed (s)')
    m.defMetric('packet_time_us', :integer, 'Time the packet was processed (us)')
    m.defMetric('packet_sent_time_s', :integer, 'Time the packet was sent (s) for
datagram-oriented protocols',
        [['MD:Iperf','MD:hasMeasurementData','MD:OneWayDelayMeasurement'],
        ['MD:OneWayDelayMeasurement','MD:OneWayDelayMeasurementValue','%value%'],
        ['MD:OneWayDelayMeasurement','MD:defaultUnit','MU:second']
    ])
    m.defMetric('packet_sent_time_us', :integer, 'Time the packet was sent (us) for
datagram-oriented protocols',
        [['MD:Iperf','MD:hasMeasurementData','MD:OneWayDelayMeasurement'],
        ['MD:OneWayDelayMeasurement','MD:OneWayDelayMeasurementValue','%value%'],
        ['MD:OneWayDelayMeasurement','MD:defaultUnit','MU:microsecond']
    ])
  }
```

Secondly, compiling just giving the directory where it is Ontology:

```
$ oml2-scaffold --oml schema.rb --ontology  ./etsi-moi-ontology/
```

Thirdly, run the semantic endpoint:

FUSEKI:

```
$ fuseki-server --update -desc ./iperf.ttl /iperf_test1
```

Where “iperf.ttl” is the Fuseki configuration file [8] where it must be defined all Fuseki parameters such as query timeout, services and database directory. And then, we run the Semantic-OML server:

```
$ oml2-server -b fuseki --fus-namespace /iperf_test1
```

VIRTUOSO OPENLINK

Where after installation with detailed steps in the webpage, it can be run only executing the following lines:

```
$ cd /…/virtuoso/database

$ virtuoso-t –f virtuoso.ini
```

Where the web frontend is at http://localhost:8090 and default username/password for administrator is dba/dba and the SPARQL endpoint at http://localhost:8890/sparql. By default this endpoint has no write rights, to grant these rights, launch isql utilty from shell and execute this line in it:

```
$ grant SPARQL_UPDATE TO “SPARQL”;
```

And then execute OML server:

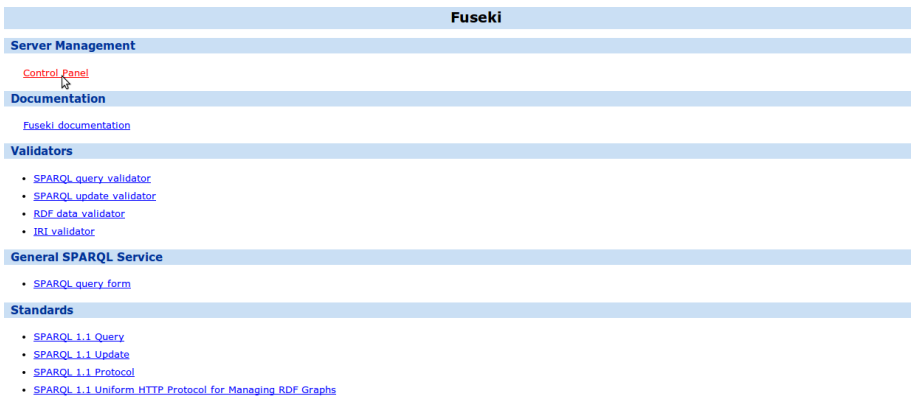
```
$ oml2-server –b virtuoso –sem-user dba –sem-pass dba –sem-host 127.0.0.1 –sem-port 8890
```

Finally, we execute Iperf server and Iperf client to collect all iperf measures:

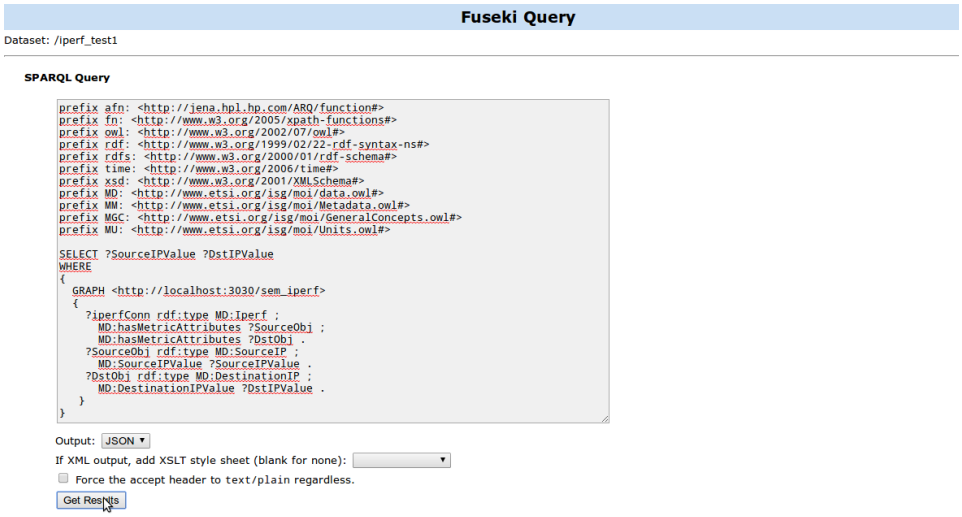
```
$ iperf --oml-id "experiment1" --oml-domain "domain1" --oml-collect "tcp:127.0.0.1:3003" --server

$ iperf --oml-id "experiment1" --oml-domain "sem_iperf" --oml-collect "tcp:127.0.0.1:3003" -c
127.0.0.1 -u -t 5
```

Then, all triples in TDB database will be stored in the directory defined in iperf.ttl or virtuoso.ini file.

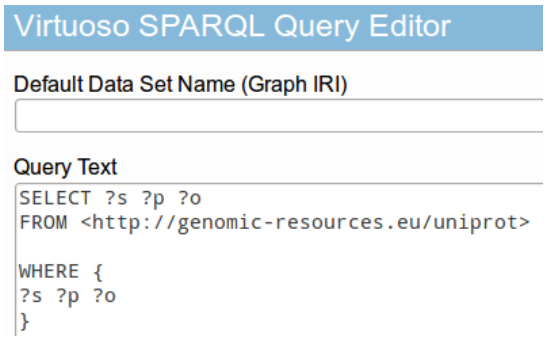
To query results in the Fuseki web service go to http://localhost:3030/, where Fuseki is running:



And in next window we can send our SPARQL query:



And if it is Virtuoso OpenLink endpoint to query results go to http://localhost:8890/sparql web portal:



Documentation

[1] "Measurement Ontology for IP traffic (MOI); Report on information models for IP traffic measurement". ETSI DSG/MOI-0010, May 2010.

[2] "Measurement Ontology for IP traffic (MOI); Requirements for IP traffic measurement ontologies development", ETSI GS MOI 002 V1.1.1, July 2012.

[3] Measurement Ontology for IP traffic (MOI); IP traffic measurement ontologies architecture," ETSI GS MOI 003 V1.1.2 January 2013.

[4] https://rubygems.org/gems/rdf-rdfxml

[5] https://rubygems.org/gems/text

[6] http://www.ict-openlab.eu/

[7] http://www.w3.org/TR/sparql11-update/

[8] http://jena.apache.org/documentation/serving_data/#fuseki-configuration-file

[9] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSBuild

C Publicación en el TridentCom

A semantic interface for OpenLab network measurement infrastructures

Jorge E. López de Vergara, Víctor Acero, Mario Poyato, Javier Aracil

High Performance Computing and Networking Research Group,
Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, Spain
jorge.lopez_vergara@uam.es

Abstract. This demo presents a semantic approach to integrate network measurement information. For this, we use a common ontology for network measurements, taking the results of the ETSI Monitoring Ontology for the Internet (MOI). This ontology allows working with a common information model, but it is also necessary to define mappings to each measurement database schema. Finally, the user can get the integrated information by distributing a semantic query among every data sources containing the monitored information.

1 Introduction

OpenLab[2] is a European project that aims at providing large scale shared experimental network facilities. OpenLab is composed of several testbeds, where each one includes monitoring tools to obtain network measurements, such as packet delays and losses, link bandwidth usage, etc. It is important for the testbed users to have an integrated view of their experiments measurements. However, each monitoring tool provides its own view of the network measurements. Most times, these measurements deal with very similar information, but represented following different structures.

2 Demonstration description

The proposed demonstration shows how the measurement information can be integrated from multiple measurement repositories. To get the information from them, a single integrated query will be needed. Using the concepts defined in the MOI ontology[3], the query will be distributed among the available repositories.

The information in the repositories is usually represented in different formats. For this reason, a mapping between each measurement repository schema and the MOI ontology concepts is defined. The researcher aiming to query network measurements will not have to know the underlying databases that contain such data, but only the MOI ontology.

To achieve the semantic integration, a system capable of querying multiple measurement repositories is needed. Users send integrated SPARQL[4] queries to an interface, which are

[2] http://www.ict-openlab.eu/

[3] http://portal.etsi.org/portal/server.pt/community/MOI

[4] http://www.w3.org/TR/rdf-sparql-query/

then translated and distributed to the multiple measurement repositories. Each SQL repository has an SPARQL endpoint provided by a D2R server[5] that maps each database table to a set of ontology concepts. Then, a mapping file is defined for each repository.

In order to obtain a better performance when translating SPARQL queries into SQL ones, the code of the D2R server has been modified to use less redundant aliases in table joins. Another modification has also been done to automatically assign super-classes of a specified class in a `ClassMap`, making the mapping process easier.

The semantic query system has a graphical interface, shown in the figure below:

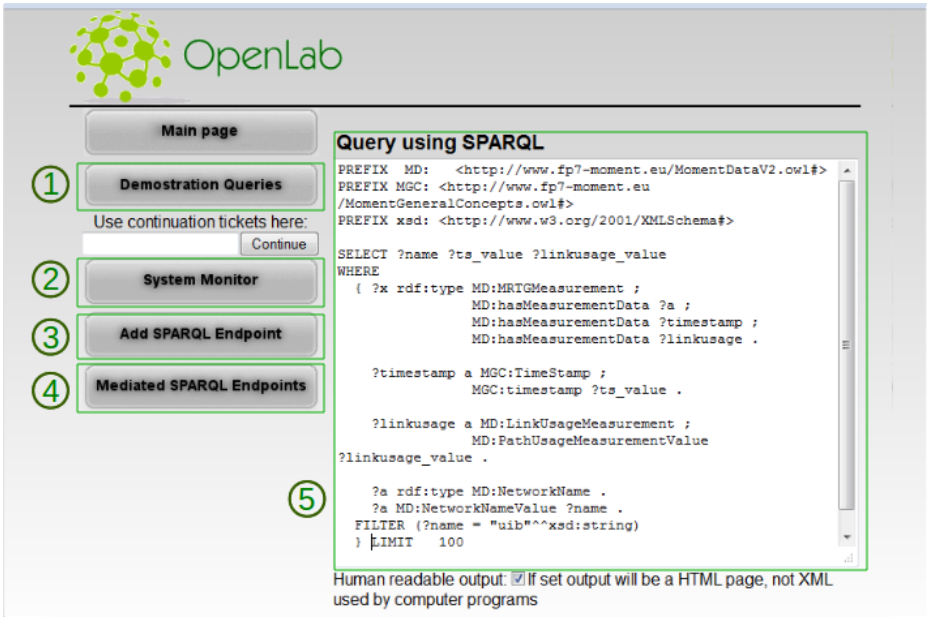


Fig. 1. OpenLab Semantic query interface.

The elements of the interface (highlighted in **Fig. 1**) are the following:

1. In “Demonstration Queries”, a set of predefined SPARQL queries can be executed.
2. In “System Monitor”, currently executing queries can be checked.
3. In “Add SPARQL Endpoint”, a new data repository can be added to the system.
4. In “Mediated SPARQL Endpoints”, a list of available data repositories can be checked.
5. This area is the core of the system, allowing a user to write a SPARQL query for the requested network measurements.

[5] http://d2rq.org/d2r-serverD2